Модуль 3. Виртуализация и контейнеризация.

Одной из самых серьезных технологических проблем в проектах разработки программного обеспечения является обеспечение совместимости различных библиотек, используемых в проекте. Для машинного обучения разработано множество open-source инструментов (библиотек, фреймворков), что является несомненным плюсом. Однако каждый из этих инструментов разрабатывается по собственной стратегии, разными командами разработчиков. Разные библиотеки развиваются и меняют версии с разной скоростью, что приводит к несовместимости различных библиотек между собой. Такие проблемы очень тяжело диагностировать, **поэтому важная часть в работе с моделями машинного обучения на всех этапах это обеспечение повторяемости результата за счет строгого контроля версий всех компонентов используемого программного обеспечения и системных настроек среды, в которой выполняется программа.** **Это достигается с помощью создания выделенных сред выполнения программ, среди которых наиболее популярными являются виртуализация ресурсов и контейнеризация приложений.** Архитектура программного обеспечения подстраивается под эти подходы, отдавая предпочтение все более популярной идеологии микросервисной архитектуры программного обеспечения, соответствующей разделению общего монолитного решения на отдельные независимые компоненты.

Этот модуль необходим для того, чтобы вы познакомились с основными технологиями и инструментами создания изолированных сред выполнения программ и научились применять эти знания для решения задач MLOps.

В результате изучения данного модуля вы сможет объяснить для чего нужны виртуализация и контейнеризация в проектах машинного обучения и использовать типовые инструменты для решения важных задач в проектах машинного обучения: обеспечения повторяемости результата в разработке и эксплуатации, обеспечения совместимости библиотек, быстрого развертывания сред разработки и эксплуатации, управления конфигурациями сред.

Темы, изучаемые в модуле:

1. Зачем необходимы изолированные среды выполнения программ и какие есть подходы для ее реализации.
2. Виртуальные машины: подходы и программное обеспечение.
3. Виртуальные среды: подходы и программное обеспечение.
4. Контейнеризация и практические аспекты работы с docker и docker-compose.

# Модуль 3. Юнит 1. Различные подходы к созданию изолированных программных сред.

*Введение:* В этом юните рассматриваются различные подходы к созданию виртуальных изолированных сред для выполнения программ.

*Содержание юнита:*

**Подход №1 “Виртуализация”**

Термин “виртуализация” стал очень популярным благодаря развивающимся технологиям облачных вычислений. Появились облачные сервисы, предоставляющие услуги доступа к виртуальным аппаратным и программным ресурсам таким образом, что для пользователя это выглядит как работа с локальным компьютером или сервером. Такой подход эффективен тем, что позволяет оптимально использовать имеющиеся аппаратные ресурсы:

* вычислительные процессоры
  + центральный процессор (CPU, Central Processing Unit),
  + графический процессор (GPU, Graphics Processing Unit),
  + тензорный процессор (TPU, Tensor Processing Unit),
* оперативную память,
* средства сетевого обмена информацией,
* средства хранения информации, жесткие и оптические диски.

При виртуализации имеющиеся аппаратные ресурсы разделяются на логические виртуальные сущности для того, чтобы создать несколько изолированных друг от друга процессов, пользующихся виртуальными ресурсами независимо друг от друга. Это позволяет оптимально распределить между изолированными процессами нагрузку на имеющиеся “физические” аппаратные мощности, избежать простоя и непроизводительного использования ресурсов, повышает рентабельность вложений в аппаратное обеспечение. **Изолированный процесс, пользующийся виртуальными ресурсами, называется виртуальная машина.**



Рисунок “Разделение аппаратных ресурсов между виртуальными машинами”

Виртуальная машина имеет свою операционную систему и работает как независимый компьютер. Кроме того, виртуальная машина имеет свои уникальные характеристики мощности процессора и объема памяти, в одних системах нужен большой объем оперативной памяти, а в других - повышенные требования к скорости вычислений. Ясно что суммарный объем “виртуальных мощностей” не может превышать мощности “физического” аппаратного обеспечения. В современных кластерах одновременно может работать много виртуальных машин, поэтому актуален вопрос их управления и мониторинга. Система, управляющая виртуальными машинами, называется **гипервизор**. Оборудование, на котором работает виртуализация, называется **хост**, его операционная система - **хостовой**. В противоположность этому, **гостевые** операционные системы это операционные системы, работающие в виртуальных машинах, пользующиеся ресурсами хоста. Гостевые операционные системы работают с эмуляцией аппаратных ресурсов.

Основная роль гипервизора состоит в координации работы виртуальных машин и физического оборудования хоста при предоставлении виртуальным машинам доступа к физическим ресурсам. Также гипервизор изолирует виртуальные машины друг от друга, чтобы они не конфликтовали за память или вычислительный ресурс.

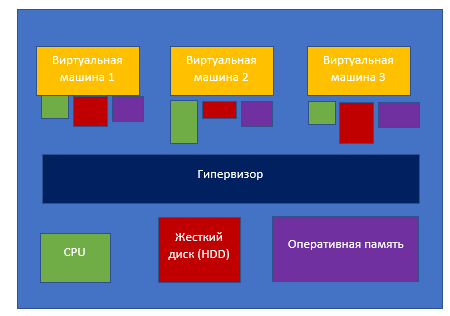


Рисунок “Роль гипервизора в координации доступа виртуальных машин к ресурсам”.

По такой схеме работают все системы виртуализации, отличаясь между собой в технических деталях и реализации. Вот, например, схема работы гипервизора Hyper-V, который входит в состав Microsoft Windows.

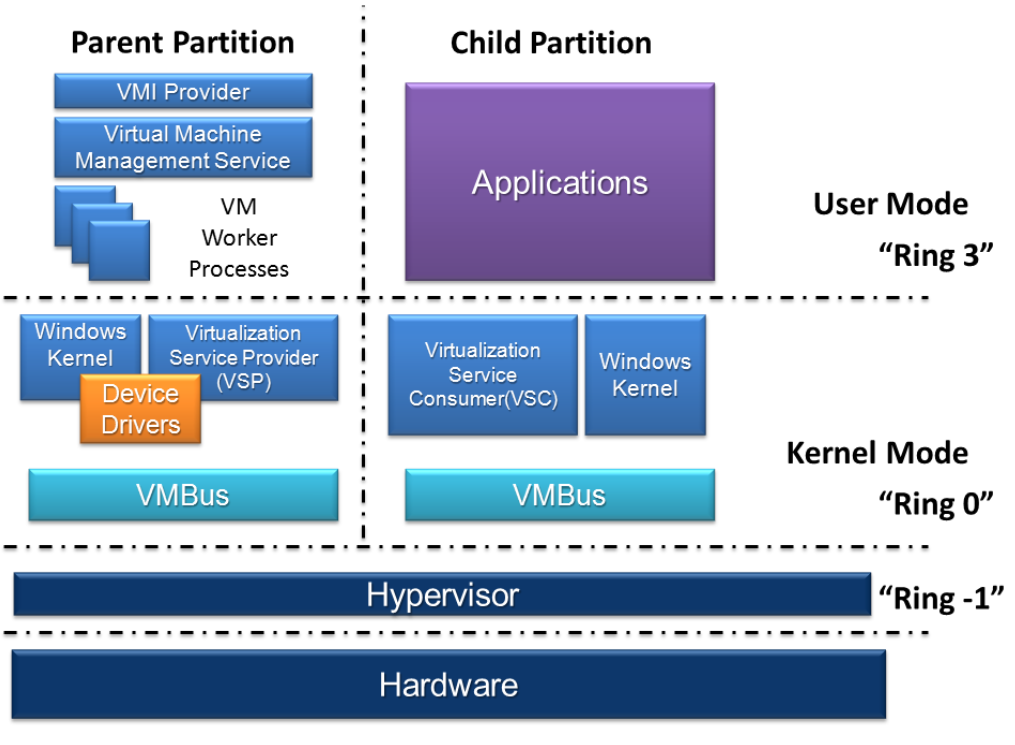


Рисунок “Архитектура работы гипервизора Hyper-V, схема с официального сайта https://docs.microsoft.com/ru-ru/virtualization/hyper-v-on-windows/about/”.

Технология виртуализации получила широкое распространение благодаря своим несомненным преимуществам:

* оптимальное использование аппаратных ресурсов
  + Один аппаратный ресурс может быть задействован в нескольких виртуальных машинах, которые будут разделять его использование между собой, что снижает время непроизводительного простоя.
  + *Пример для MLOps: как вы уже наверное знаете, популярный вычислительный ресурс GPU в машинном обучении стоит очень дорого, однако применяется главным образом для обучения моделей. В то время как одна модель уже обучена и проходит тестирование, ресурс GPU может быть использован для обучения другой модели.*
* дополнительные возможности обеспечения безопасности
  + Виртуализация позволяет создать копию рабочей системы, чтобы быстро переключиться на вторую систему при выходе из строя первой. Кроме того, виртуальная машина выглядит идентично обыкновенной системе, однако злоумышленник, получивший доступ к виртуальной машине, не сможет вывести из строя основную, хостовую систему.
  + *Пример для MLOps: в случае возникновения неполадок при работе можно переключиться на резервную систему, сократив время простоя.*
* быстрое развертывание:
  + Можно создать типовые шаблоны и политики для создания виртуальных машин и их настройки, единообразный подход позволяет упростить и ускорить процедуру развертывания решения. Для унифицированной конфигурации проще проанализировать технологические угрозы и есть возможность создать единые политики безопасности и контролировать их выполнение.
  + *Пример для MLOps: для рабочей конфигурации основных элементов можно создать правила развертывания и настройки, сведя к минимуму время на развертывание системы.*

**Как видите, технологии виртуализации поддерживают тренд в разработке и использовании программного обеспечения, который мы обсуждали в предыдущих модулях, обеспечивая скорость выполнения операций и надежность.**

Примеры виртуальных машин:

* VirtualBox компании Oracle (<https://www.virtualbox.org/wiki/Downloads>), бесплатный инструмент для виртуализации серверов,
* VMware (https://www.vmware.com), обеспечивает виртуализацию серверов, рабочих столов (desktops), хранилищ данных,
* Hyper-V (https://docs.microsoft.com/ru-ru/virtualization/hyper-v-on-windows/about/), входит в состав ОС Microsoft Windows, предоставляет решения по виртуализации серверов и рабочих компьютеров,
* Citrix, виртуализация для серверов, рабочих компьютеров и приложений.

Список инструментов для виртуализации достаточно широкий, например, в него также входят решения IBM LPAR, Xen, KVM, Bhyve. В конечном счете специалисты DevOps выбирают наиболее оптимальные инструменты, руководствуясь необходимыми функциями, производительностью, стоимостью решения.

С другой стороны, как и любая новая полезная технология, виртуализация порождает новые проблемы с безопасностью. Например, если злоумышленник взломает гипервизор, то он может получить контроль над всеми виртуальными машинами и их операционными системами. Эту проблему будет сложно диагностировать с помощью анализа сетевого трафика, так как гипервизоры обеспечивают взаимодействие виртуальных машин друг с другом без участия физической сети. Поэтому необходимо применять продукты для безопасности виртуализации, способные сканировать и исправлять виртуальные машины, зараженные вредоносным кодом, шифровать все виртуальные диски виртуальных машин и контролировать доступ к ним.

Необходимо отметить, что кроме рассмотренной виртуализации серверов, рабочих компьютеров, процессоров и систем хранения данных еще существуют технологии виртуализации сетей, приложений, отдельных компонентов программной системы или даже больших комплексных решений, например ЦОДов. Основные преимущества подхода точно такие же, как мы рассмотрели, быстрая настройка и конфигурирование, простота в эксплуатации, гибкое управление и надежность.

**Подход №2 Контейнеризация**

Теперь давайте рассмотрим другой популярный подход для создания изолированной среды выполнения программы, который называется **контейнеризация**. Виртуальные машины и контейнеры принципиально отличаются друг от друга, хотя и решают похожие задачи. При виртуализации необходимо полностью создавать систему, повторяющую полноценный рабочий компьютер или сервер, включая операционную систему и системные службы, даже если мы создаем среду для выполнения всего одной программы. Конечно же это не производительно, с точки зрения ресурсов, создает избыточность и дополнительные накладные расходы. **Контейнеризация,** ярким представителем которой является docker, **решает эту проблему с использованием контейнеров для упаковки прикладного программного обеспечения, которые обращаются к единому ядру хостовой операционной системы, запуская для себя только те библиотеки или функции, которые нужны для контейнера и приложения в нем.** Это позволяет существенно уменьшить накладные расходы на создание и использование контейнера и повысить общую производительность. То есть, **контейнерная виртуализация — это виртуализация на уровне хостовой операционной системы, которая позволяет запускать изолированные виртуальные системы на одном физическом узле, но не позволяет запускать операционные системы с ядрами, отличными от типа ядра базовой операционной системы.** При таком подходе не существует отдельного слоя гипервизора, вместо этого сама хостовая операционная система отвечает за разделение аппаратных ресурсов между несколькими гостевыми системами (контейнерами) и обеспечивает их независимость.

Идея контейнеризации появилась достаточно давно и у современных популярных инструментов есть достаточно много предшественников: FreeBSD Jail (2000), Virtuozzo Containers (2000), Solaris Containers (2005), OpenVZ (2005), [LXC](https://ru.wikipedia.org/wiki/LXC) (2008), iCore Virtual Accounts (2008). Появившиеся и развивавшиеся в виде стартапов продукты Docker (2013) и Kubernetes (2014) быстро завоевали большую популярность и в настоящее время являются основными средствами контейнеризации. Например, docker, ставшим стандартом “де-факто”, настолько популярен, что появился даже специальный термин **“докеризация”**. Мы будем рассматривать этот инструмент в следующих юнитах данного модуля. Когда контейнеров много появляются дополнительные издержки на их синхронизацию между собой, актуальным становится вопрос резервирования. Для управления группой контейнеров используются инструменты docker-compose и kubernetes.

Для полноты картины необходимо упомянуть еще инструмент OpenStack (<https://www.openstack.org>), представляющий собой набор open-source проектов программного обеспечения. OpenStack может быть использован для создания инфраструктурных облачных сервисов и облачных хранилищ. Все проекты OpenStack распространяются под лицензией Apache License. Это очень большой проект, включающий множество инструментов, разрабатываемых большим количеством компаний-разработчиков. Перечень отдельных модулей OpenStack очень обширный. OpenStack не является популярным инструментом в MLOps, поскольку задачи в проектах машинного обучения эффективнее решаются другими инструментами. Однако OpenStack набирает популярность в индустрии разработки программного обеспечения, поэтому рекомендуем вам познакомиться с этим инструментом самостоятельно.

В следующих юнитах рассмотрим инструменты для создания виртуальных машин, программных виртуальных сред и контейнеров. Инструментов для виртуализации и контейнеризации достаточно много. Мы в данном модуле ограничимся рассмотрением инструмента VirtualBox для виртуализации и docker и docker-compose для контейнеризации.

*Тест*

1. Отметьте гипервизоры (0.25)
   1. **Hyper-V**
   2. docker
   3. **KVM**
   4. **VMWare**
2. Отметьте инструменты для контейнеризации (0.25)
   1. **docker**
   2. **kubernetes**
   3. containers
   4. VMWare
3. Отметьте задачи, которые решает виртуализация в проектах машинного обучения (0.25)
   1. дает возможность делать качественные визуализации
   2. **ускоряет развертывание проекта с заданными характеристиками**
   3. **обеспечивает повторяемость результатов работы модели**
   4. позволяет обрабатывать большие объемы информации
4. Отметьте положительные стороны использования виртуальных машин (0.25)
   1. **возможность создания и использования унифицированных типовых конфигураций и политик для виртуальных машин**
   2. существенное ускорение производительности оборудования
   3. **оптимизация расходов на аппаратное обеспечение**
   4. гарантия качества работы виртуальной машины компанией-разработчиком

*Итоги/выводы*

В этом юните вы изучили ключевые технологии для создания виртуальных изолированных сред выполнения программы: виртуализацию и контейнеризацию. Эти технологии соответствуют общему тренду использования микросервисной архитектуры при создании программного обеспечения, ускоряют операции и процессы в проекте.

Наиболее популярные инструменты для виртуализации: VirtualBox Oracle, VMWare, KVM, Hyper-V.

Практически без конкурентов в настоящее время инструменты контейнеризации: docker/docker-compose и kubernetes.

В следующих юнитах мы подробнее рассмотрим создание виртуальных машин с VirtualBox и контейнеризацию с docker/docker-compose.

# Модуль 3. Юнит 2. Виртуализация с использованием VirtualBox.

*Введение:* В этом юните рассматривается задача виртуализации с использованием популярного бесплатного инструмента VirtualBox компании Oracle. С использованием простых шагов, подробно описанных в этом юните, вы сможете самостоятельно создавать виртуальные машины.

*Содержание юнита:*

Инструмент для виртуализации VirtualBox разработан компанией Oracle и распространяется бесплатно. Инструкции по установке и дистрибутивы для скачивания можно найти здесь:

<https://www.virtualbox.org/wiki/Downloads>

Также для работы вам понадобится образ для операционной системы, которая будет работать в виртуальной машине. В этом юните мы будет использовать Ubuntu, образ которой можно скачать здесь: <https://ubuntu.com/download/server>. Необходимо учитывать, что это очень объемный образ, поэтому для экспериментов с созданием виртуальных машин можно также воспользоваться образом более “легкой” операционной linux системы alpine, скачать которые можно здесь: https://alpinelinux.org/downloads.

Для разнообразия в этом юните мы будем изучать практическое применение инструментов в операционной системе Windows 64-бит, надо внимательно выбрать подходящую программу для установки VirtualBox на сайте. Кроме установщика самой программы понадобится еще установщик для пакета расширений VirtualBox, его можно скачать там же.

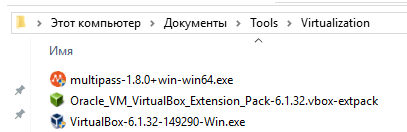
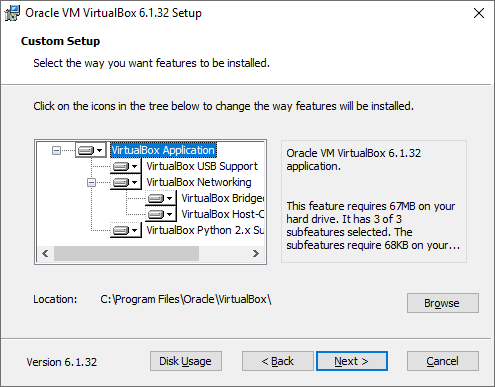
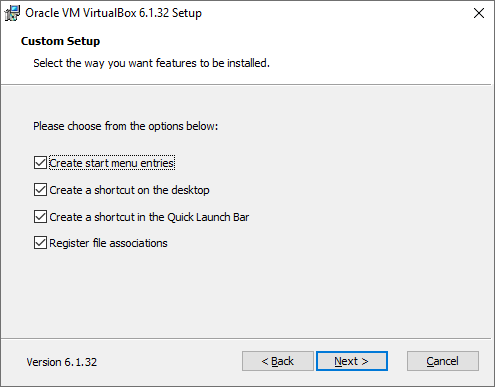


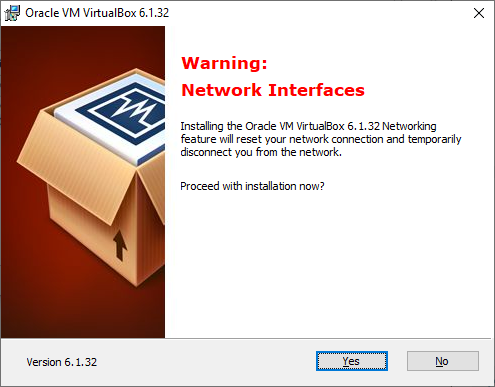
Рисунок “Скачанные дистрибутивы для установки VirtualBox”

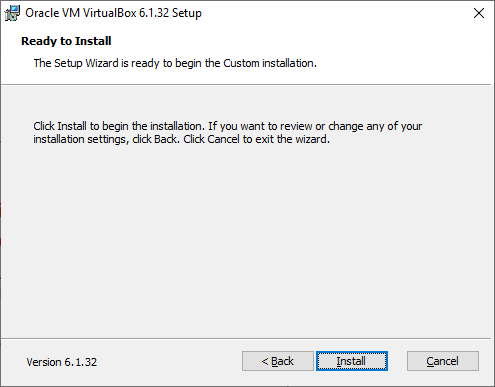
После запуска исполняемого файла “VirtualBox….-Win.exe” вы будете следовать обычному сценарию установки программного обеспечения в Windows. Далее приведены скриншоты отдельных этапов установки.

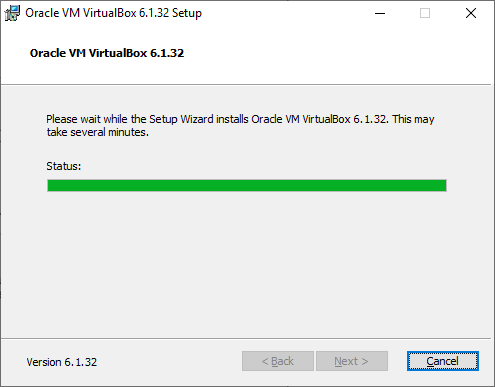


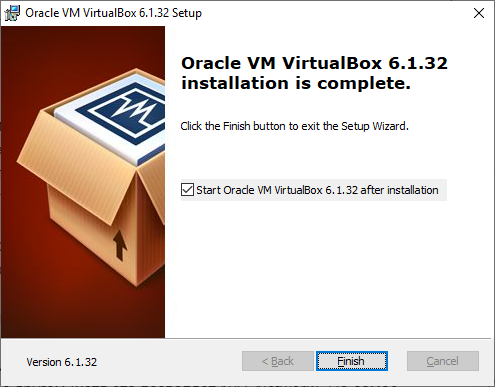


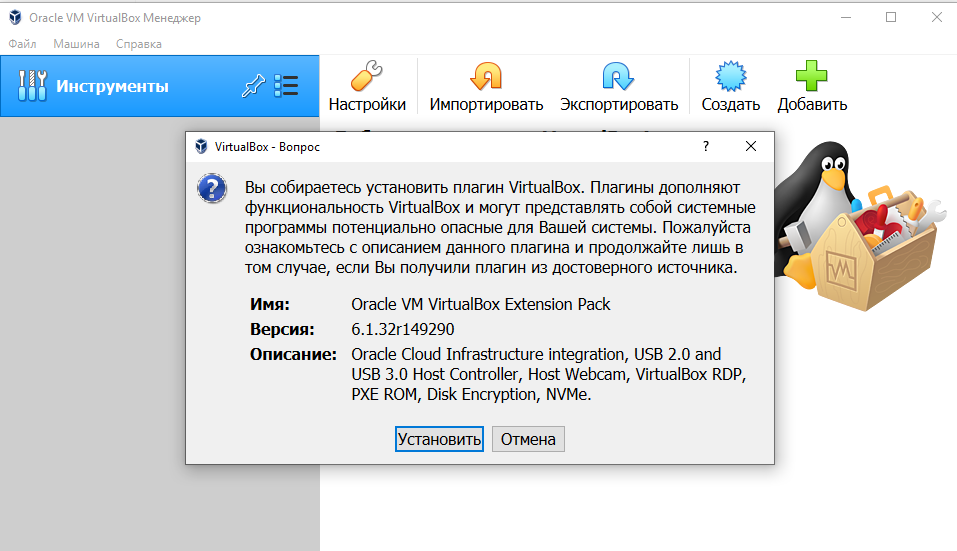


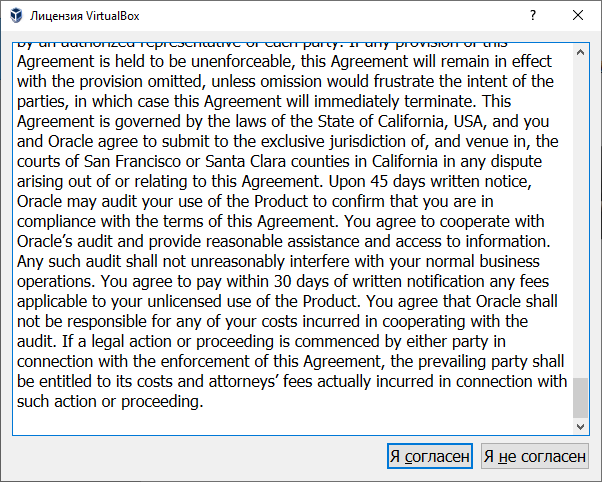


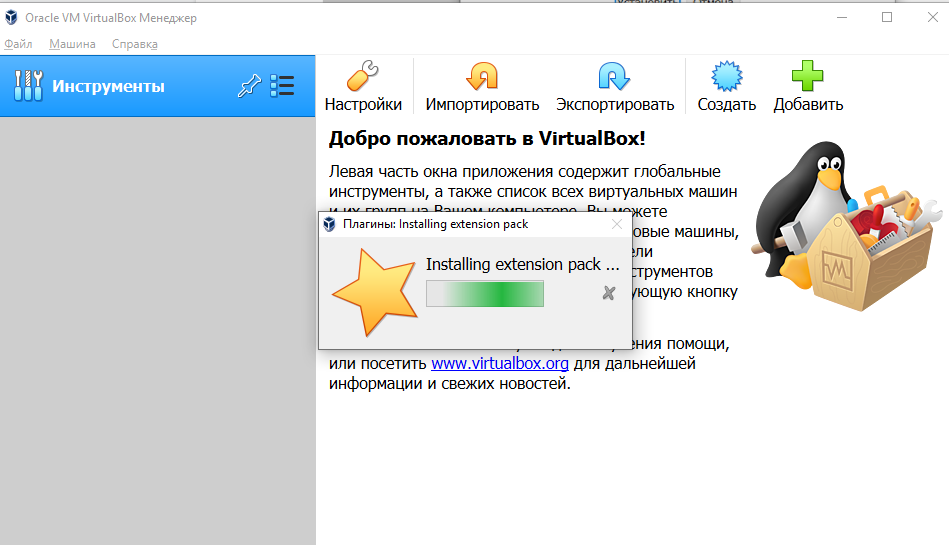


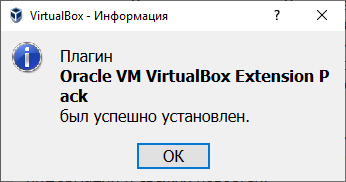




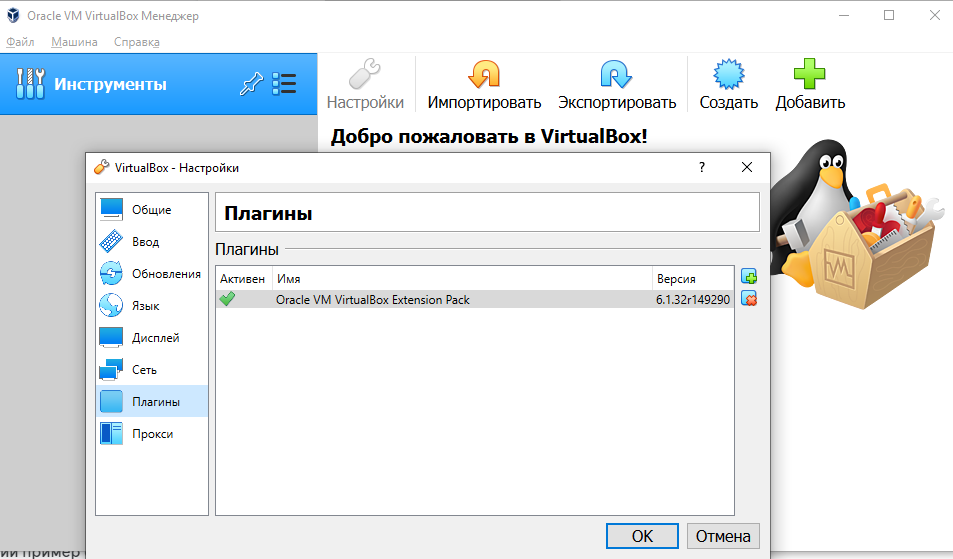








После установки в меню **Файл -> Настройки -> Плагины** (или, в англоязычном варианте, **File -> Preferences -> Extensions**) можно перейти в настройки и убедиться, что Extension Pack установлен.



В VirtualBox каждая виртуальная машина может иметь до четырех сетевых адаптеров, а каждый такой адаптер имеет определённый режим работы:

* NAT (Network Address Translation): позволяет виртуальным машинам получать доступ в интернет через хост, но не позволяет им взаимодействовать друг с другом, IP в этом случае назначается динамически, виртуальные машины доступны из внешней сети, но внешняя сеть видит только хост машину,
* NAT Network: то же, что и NAT, но виртуальные машины могут взаимодействовать между собой через внутреннюю сеть,
* Bridged: виртуальная машина имеет свой собственный статический IP адрес и доступна из внешней сети напрямую,
* Host-only: создается одна общая сеть между хостом и всеми виртуальными машинами, каждой из которых можно назначить статический IP адрес, требует создания виртуального сетевого адаптера на хосте.

**Эти режимы отвечают за способ взаимодействия виртуальных машин с хостом и между собой, а также будет ли у них доступ в интернет и смогут ли другие устройства в сети хоста взаимодействовать с этими виртуальными машинами.**

Виртуальный адаптер сети хоста доступен в меню Файл -> Менеджер сетей хоста.

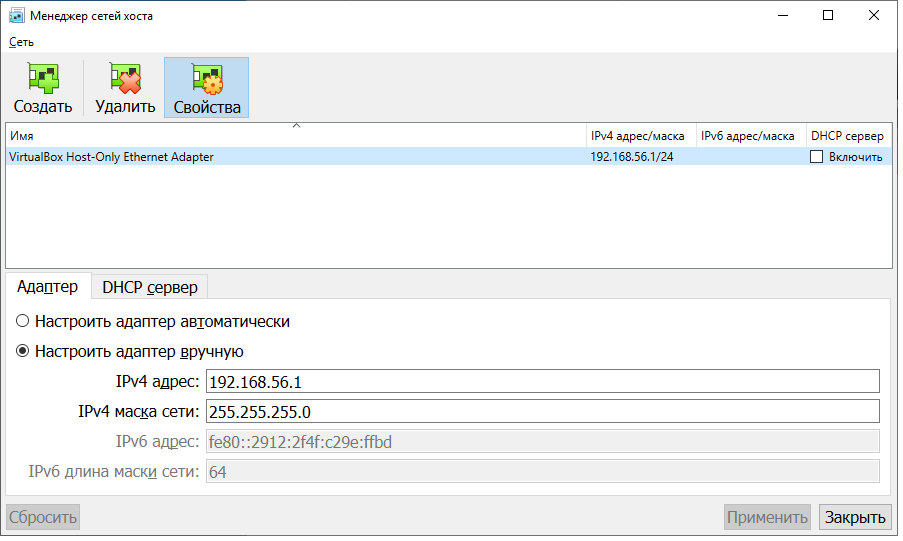


Рисунок “Менеджер сетей хоста VirtualBox”.

Если требуется отключить динамическое назначение IP адресов для виртуальных машин, то необходимо отключить эту опцию в параметре “DHCP сервер”. Поле “IPv4 адрес” задает IP адрес хоста. Мы можем назначать IP адреса виртуальным машинам из одной подсети с хостом, подсеть задается маской в поле “IPv4 маска сети”. После изменения адреса сетевого адаптера необходимо обязательно перезагрузить компьютер, т.к. без этого виртуальные машины не будут запускаться.

Давайте создадим шаблонную виртуальную машину. Для этого в пункте меню “Машина” необходимо выбрать раздел “Создать”.

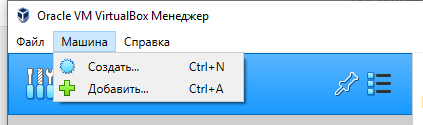


Рисунок “Создание виртуальной машины”

После этого можно задать имя виртуальной машины, указать ее тип и версию.

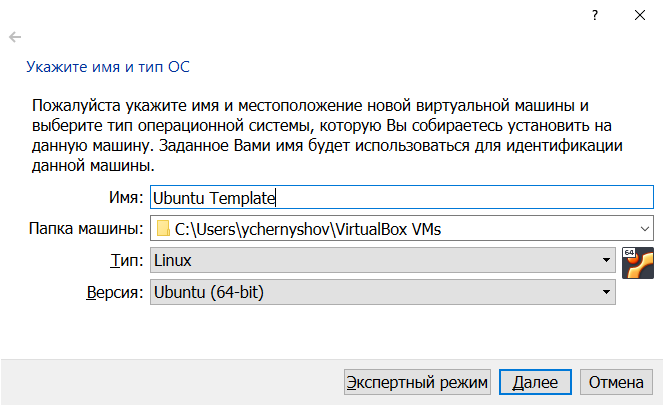


Рисунок “Указание параметров виртуальной машины”

Также можно перейти в расширенные настройки параметров виртуальной машины и изменить такие параметры как объем памяти и способ организации виртуального жесткого диска.

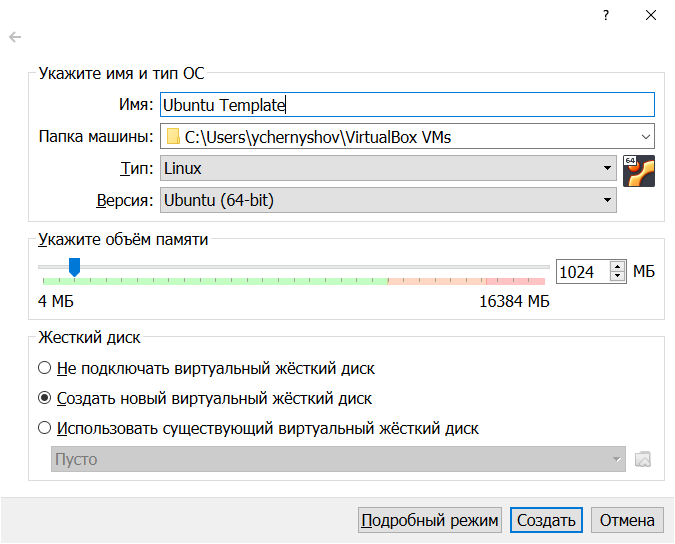


Рисунок “Расширенные настройки виртуальной машины”.

После этого создаем для виртуальной машины новый виртуальный жёсткий диск в формате VDI (VirtualBox Disk Image) с форматом хранения “Динамический виртуальный женсткий диск” и размером 10Гб.

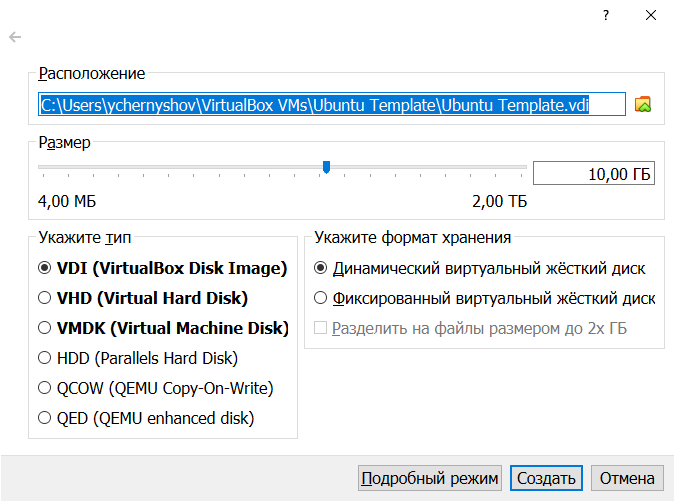


Рисунок “Параметры виртуального жесткого диска”

После этого наша виртуальная машина готова.

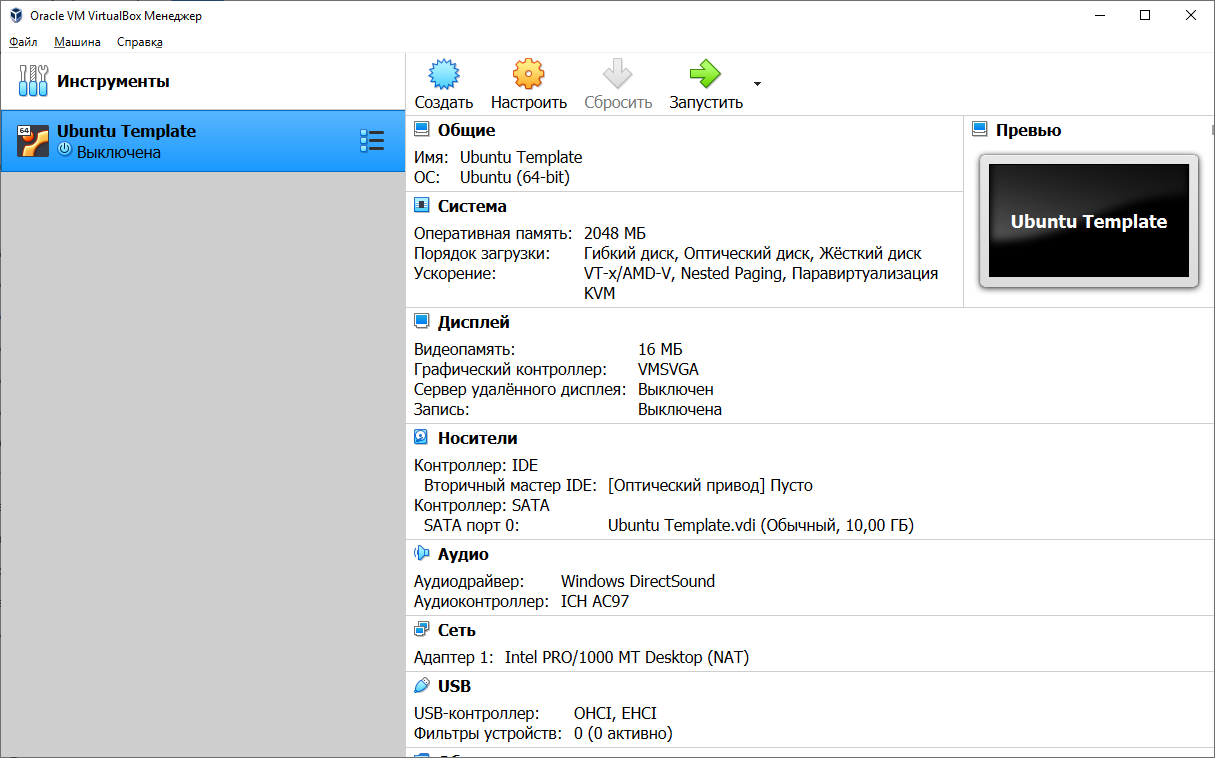


Рисунок “Переметры созданной виртуальной машины VirtualBox”.

У созданной виртуальной машины можно теперь менять параметры, например, изменить количество процессоров

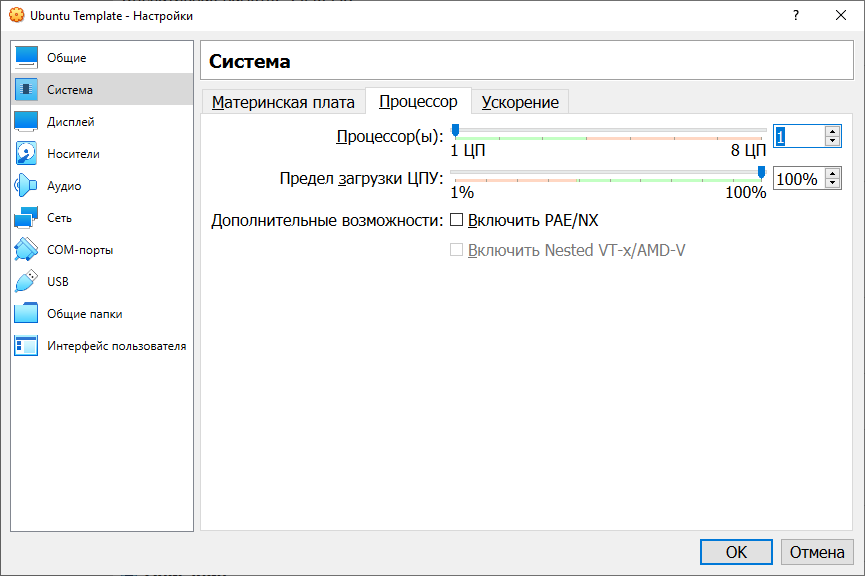
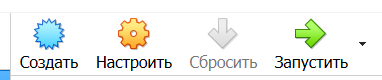


Рисунок “Настройки параметров виртуальной машины, изменение количества процессоров”.

Конечно же, виртуальная машина пока не может эксплуатироваться, хотя и обладает уже виртуальными ресурсами - процессором, оперативной памятью, жестким диском, сетевым адаптером. Для работы необходимо использовать установочный образ, с которого в виртуальную машину будет загружена гостевая операционная система. *Один из способов указать виртуальной машине откуда брать загрузочные данные, это создать виртуальный оптический диск и прикрепить к нему образ операционной системы, скачанный с официального сайта (например, в формате ISO).* Для этого в панели управления виртуальными машинами надо для рабочей виртуальной машины выбрать пункт “Настроить”



после чего появится следующее окно для настроек

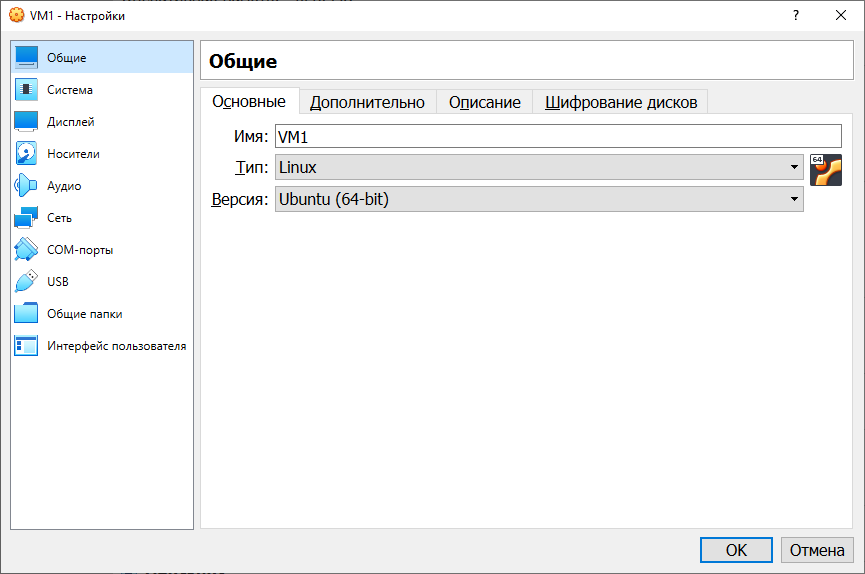
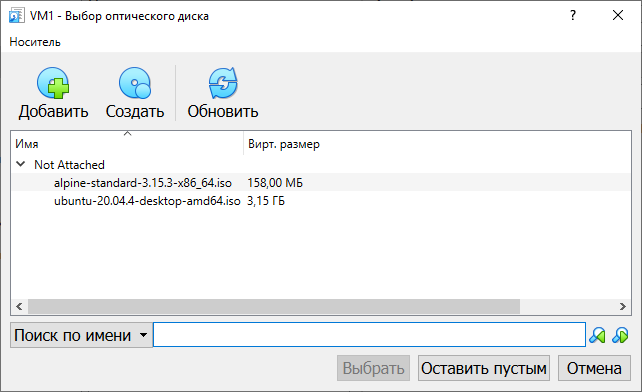
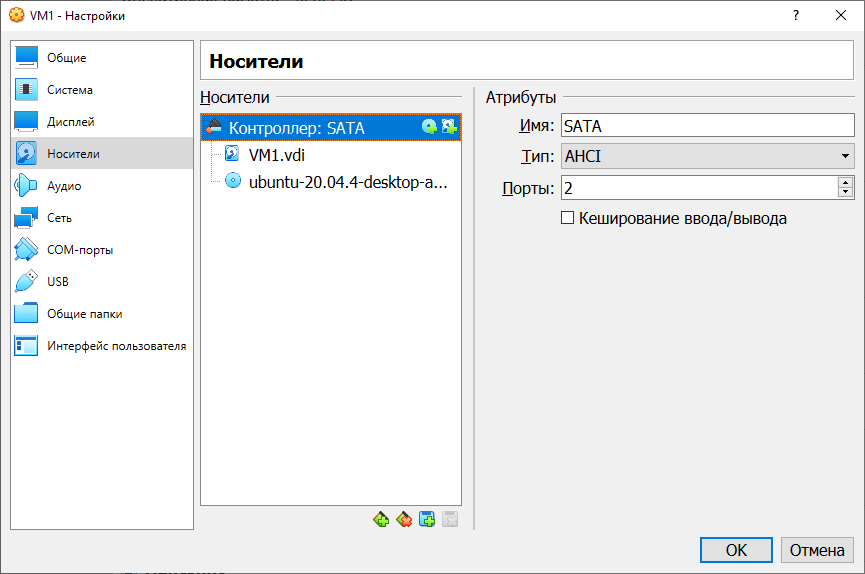


Рисунок “Настройки виртуальной машины”

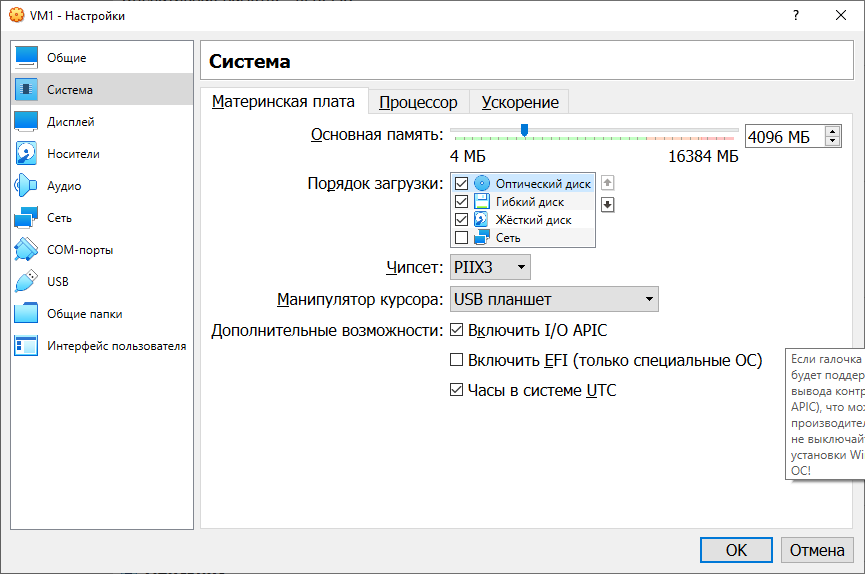
Сначала в разделе “Носители” надо добавить оптический диск



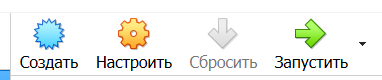
Он должен появиться в перечне дисков



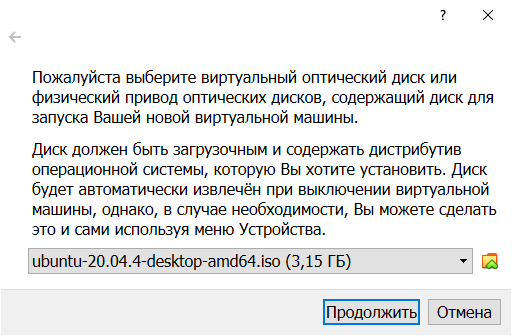
После этого в разделе “Система” меняем порядок загрузки, делаем оптический диск первым



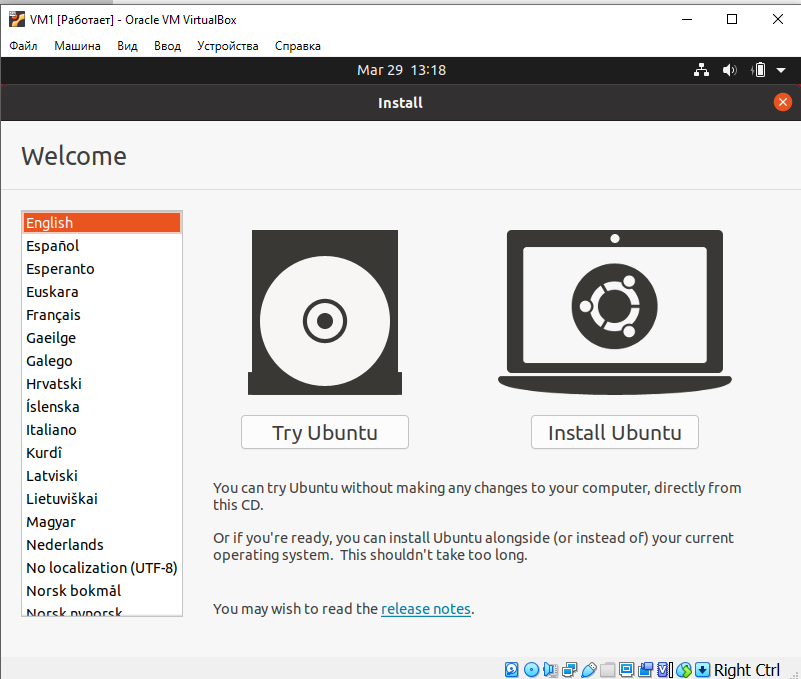
После этого нажимаем кнопку “Запустить”



и в появившемся окне надо выбрать виртуальный загрузочный диск



После этого система начнет использовать диск с загрузчиком операционной системы, в частности, вы увидите диалог для установки Ubuntu



*Полезная информация: если вы нажмете курсором мыши на окно виртуальной машины, то произойдет перехват управления и передача всех управляющих сигналов от мыши и клавиатуры к виртуальной машине. То есть мышь и клавиатура не будут работать в домашней операционной системе. Поначалу это может удивлять и пугать. Вернуть управление можно нажатием специальной клавиши, в большинстве случаев это правая клавиша Ctrl.*

Теперь у вас на одном компьютере может быть много различных операционных систем, работающих независимо и изолированно друг от друга.

Дополнительно к информации о VirtualBox еще отметим инструмент Vagrant, свободное и открытое программное обеспечение для создания и конфигурирования виртуальной среды разработки. Vagrant является оберткой для программного обеспечения виртуализации, например, VirtualBox, и средств управления конфигурациями, таких как Chef, Salt и Puppet.

Установка vagrant

**sudo apt install virtualbox**

**sudo apt install vagrant**

**vagrant –version**

*Тест*

1. Как можно получить официальный дистрибутив VirtualBox? (0.25)
   1. **скачать установщик с сайта производителя**
   2. использовать утилиту pip install
   3. использовать консольную команду virtualbox install
   4. эта утилита входит по умолчанию во все стандартные операционные системы, ее не надо скачивать и устанавливать
2. сколько виртуальных сетевых адаптеров может быть связано с виртуальной машиной в VirtualBox? (0.25)
   1. 1
   2. 2
   3. 3
   4. **4**
3. сколько операционных систем может работать в виртуальной машине? (0.25)
   1. **1**
   2. 2
   3. 3
   4. 4
4. как вернуть перехваченное управление клавиатурой и мышью из гостевой операционной системы обратно в операционную систему хоста? (0.25)
   1. написать письмо в техническую поддержку производителя
   2. **нажать правую клавишу ctrl**
   3. перезагрузить компьютер
   4. нажать ctrl+alt+del.

*Итоги/Выводы*

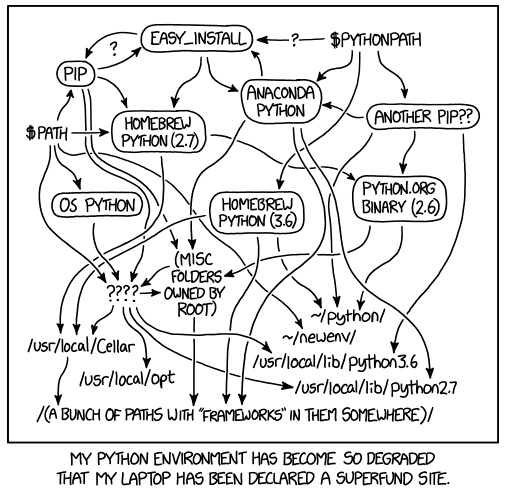
В этом юните вы научились устанавливать программное обеспечение VirtualBox Oracle и использовать его для создания виртуальных машин. В качестве примера мы разобрали создание виртуальной машины с операционной системой Ubuntu.

# Модуль 3. Юнит 3. Виртуальные окружения.

*Введение:* Виртуализация и контейнеризация решают задачу создания среды выполнения программы на уровне аппаратного обеспечения или ядра операционной системы хоста, что требует достаточно много ресурсов. На практике для отдельных задач в проекте разработки программного обеспечения бывает достаточно в операционной системе создать виртуальное окружение, без создания новой виртуальной машины или контейнера. В этом случае изоляция будет не очень сильной, однако это позволяет изолировать среду для уникального набора используемых утилит, настроек, библиотек, например, python. Такой механизм называется **виртуальное окружение** (virtual environment). Виртуальные окружения активно применяются, в том числе в проектах машинного обучения, в сочетании с виртуализацией и контейнеризацией. В этом юните мы более подробно рассмотрим вопросы создания и использования виртуальных окружений.

*Содержание юнита:*

Виртуальное программное окружение отличается от рассмотренных виртуальных машин и контейнеров тем, что не содержит операционную систему. **Основное назначение виртуального окружения состоит в создании изолированной конфигурации программного обеспечения с определенным зафиксированным набором библиотек определенных версий.** Это позволяет быстро повторить нужную конфигурацию и избежать проблем актуальности версий и совместимости библиотек между собой. Вот иллюстрация того какие боли могут возникнуть в проекте с использованием python



https://xkcd.com/1987/

Виртуальное окружение позволяет зафиксировать работающие конфигурации используемых библиотек. При необходимости можно быстро переключиться в необходимое виртуальное окружение и запустить прикладную программу в этом уникальном сочетании библиотек, настроек, конфигураций. Это позволяет быстрее повторить и локализовать проблему. При работе с python проектами, а в машинном обучении python является наиболее распространенным инструментом, подход к созданию целевой конфигурации виртуального окружения состоит в создании и запуске этого виртуального окружения и установки необходимых версий библиотек в это окружение, например, с помощью популярной **утилиты pip** (https://pypi.org/project/pip/).

Вот основные команды при работе с pip в командной строке Windows или в терминале Ubuntu

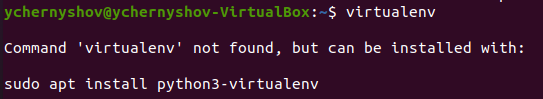
|  |  |
| --- | --- |
| Команда | Описание |
| pip help | Справка по командам |
| pip search “имя пакета” | Поиск пакета |
| pip show “имя пакета” | Информация об пакете |
| pip install “имя пакета” | Установка пакета |
| pip uninstall “имя пакета” | Удаление пакета |
| pip list | Список установленных пакетов |
| pip install -U | Обновление пакета |

Если виртуальные окружения не используются, то во время установки пакета полезно использовать дополнительно ключ --user, устанавливая пакет локально только для текущего пользователя.

Существует множество утилит для создания виртуального окружения в python, вот наиболее популярные из них: virtualenv, venv, conda, poetry. Давайте разберем работу с ними подробнее. *Кстати, для экспериментов вы можете использовать виртуальную машину с гостевой операционной системой Ubuntu, создание которой мы разобрали в предыдущем юните.*

1. **virtualenv**

По умолчанию в Ubuntu утилита virtualenv отсутствует, необходимо устанавливать.



Установка осуществляется командой

**sudo pip install python3-virtualenv**

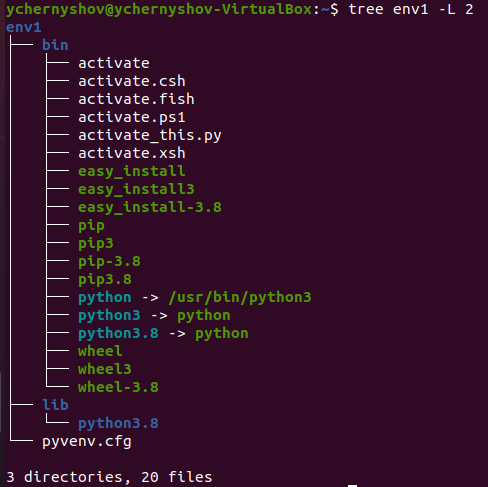
После этого можно пользоваться virtualenv



Вот основные команды при работе с VirtualEnv в командной строке Windows и в терминале Ubuntu

|  |  |
| --- | --- |
| mkvirtualenv “имя окружения” | Создать новое виртуальное окружение. Создается папка с именем “имя окружения”, содержащая всю необходимую для работы виртуального окружения информацию. |
| workon | Получить список окружений |
| workon “имя окружения” | Изменить используемое виртуальное окружение |
| deactivate | Выйти из виртуального окружения |
| rmvirtualenv “имя окружения” | Удалить виртуальное окружение |

В рабочей папке виртуального окружения после создания формируется следующая структура каталогов



1. **venv**

Использование утилиты venv аналогично использованию virtualenv. Создать виртуальное окружение можно следующим образом:

**python3 –m venv <имя папки>**

**source bin/activate**

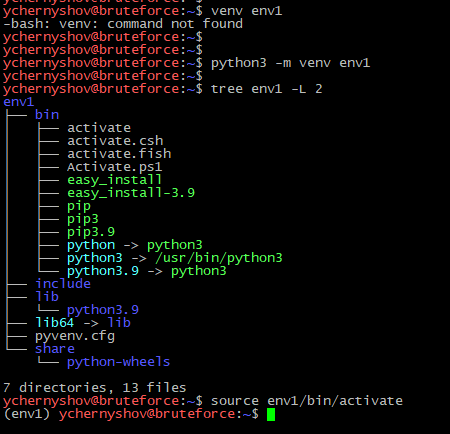
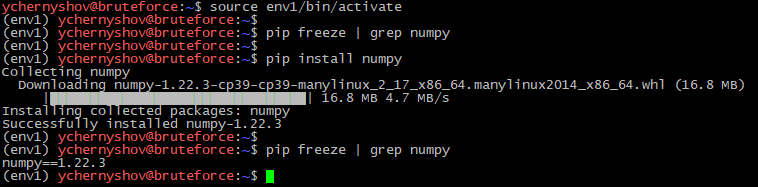
****

Рисунок “Активация виртуального окружения с помощью venv”

После выполнения скрипта activate при успешной активации виртуальной среды вы увидите соответствующий промптер перед курсором.

Теперь можно устанавливать требуемые версии библиотек программного обеспечения.



Деактивировать виртуальное окружение можно командой

**deactivate**

1. **conda**

Anaconda является еще одним популярным инструментом для организации работы над python проектам. В том числе поддерживается возможность создания виртуальных сред. Скачать дистрибутив можно здесь <https://www.anaconda.com/products/distribution>.

Виртуальная среда создается аналогично предыдущим инструментам, при этом создается папка со всем необходимым содержимым.

**conda create --name “имя окружения” python=3.6**

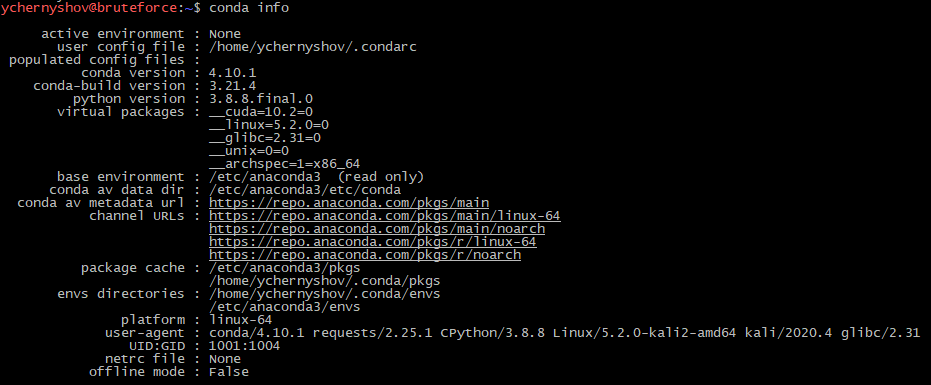
Активировать виртуальное окружение можно с помощью команды с консоли

**conda activate “имя окружения”**

Далее можно управлять содержанием виртуального окружения, добавлять нужные библиотеки

**conda install numpy pandas**

У утилиты conda есть много опций, их можно увидеть через справку. Например, можно воспользоваться командой conda info для получения подробной информации о текущей конфигурации

****

1. **poetry**

Poetry это хорошая альтернатива pip, которая позволяет отказаться от requirements.txt в пользу более гибкой настройки проекта. Благодаря poetry можно в любой момент посмотреть информацию о зависимостях любого пакета, гибко настраивать версии и обмениваться poetry.lock файлами с уже заготовленным списком версий пакетов. Установщик pip хранит данные о зависимостях в файле requirements.txt, а poetry хранит информацию в файле pyproject.toml, однако, в случае с pip, в его файле хранится только список зависимостей с описанием версий, а в .toml хранится вся основная информация о проекте, что очень удобно, так как все данные собраны в одном месте. Кроме того poetry также может управлять окружением проекта.

Установка осуществляется с помощью команды

**pip install poetry**

Главный файл для poetry - это pyproject.toml. Все данные о проекты должны быть записаны в нём. При установке пакетов poetry берёт данные из этого файла и формирует файл с зависимостями poetry.lock (если уже есть готовый файл poetry.lock, то данные будут браться из него). Toml файл состоит из нескольких блоков, каждый из которых имеет свои особенности, рассмотрим данные блоки:

[tool.poetry] - содержит основную информацию о проекте, такую как:

* name - имя проекта
* version - версия проекта
* description - описание проекта
* license - лицензия проекта
* authors - список авторов проекта в формате name <email>
* maintainers - список менторов проекта формате name <email>
* readme - readme файл проекта в формате README.rst или README.md
* homepage - URL сайта проекта
* repository - URL репозитория проекта
* documentation- URL документации проекта
* keywords - список ключевых слов проекта (макс: 5)
* classifier - список PyPI классификаторов

[tool.poetry.dependencies] - содержит описание всех зависимостей проекта. Каждая зависимость должна иметь название с указанием версии, также присутствует возможность скачать проекта с github с указанием ветки/версии/тэга, например:

* requests = "^2.26.0"
* requests = { git = "[https://github.com/requests/requests.git"](https://github.com/requests/requests.git%22) }
* requests = { git = "[https://github.com/kennethreitz/requests.git"](https://github.com/kennethreitz/requests.git%22), branch = "next" }
* numpy = { git = "[https://github.com/numpy/numpy.git"](https://github.com/numpy/numpy.git%22), tag = "v0.13.2" }

[tool.poetry.scripts] - в данном разделе можно описать различные сценарии или скрипты, которые будут выполняться при установке пакетов или при запуске приложения. Например:

* poetry = 'poetry.console:run'
* main-run = 'new\_proj.main:run' (после чего достаточно запустить poetry main-run и будет выполнен запуск функции run в файле new\_prof/main.py)

[tool.poetry.extras] - в данном блоке описываются группы зависимостей, которые можно устанавливать отдельно:

[tool.poetry.dependencies]

* psycopg2 = { version = "^2.7", optional = true }
* pymysql = { version = "1.0.2", optional = true }

[tool.poetry.extras]

* mysql = ["pymysql"]
* pgsql = ["psycopg2"]

Далее зависимости можно установить двумя способами:

**poetry install --extras "mysql pgsql"**

**poetry install -E mysql -E pgsql**

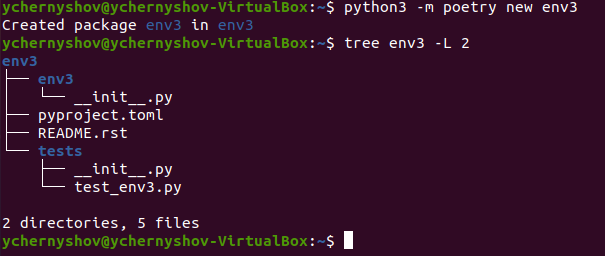
[tool.poetry.urls] - помимо основных URL, указанных в [tool.poetry], можно указывать свои URL:

* "Bug Tracker" = "[https://github.com/python-poetry/poetry/issues"](https://github.com/python-poetry/poetry/issues%22)

Чтобы создать новый проект с помощью poetry, достаточно выполнить

**poetry new <название папки с проектом>**

После чего создастся папка с названием вашего проекта, в этой папке будет лежать файл pyproject.toml.



Чтобы установить зависимости проекта достаточно выполнить команду:

**poetry install**

Чтобы добавить новую библиотеку достаточно выполнить:

**poetry add numpy**

Чтобы удалить зависимость достаточно выполнить:

**poetry remove numpy**

Чтобы посмотреть зависимости проекта достаточно выполнить:

**poetry show**

Также poetry содержит другие команды, с которыми можно ознакомиться в документации.

Итак, мы рассмотрели различные инструменты для создания виртуальных окружений. Этих инструментов достаточно много и вопрос выбора конкретного инструмента зависит от навыков и квалификации специалиста DevOps/MLOps. Более надежными считаются инструменты venv и virtualenv, поскольку они появились раньше и авторы успели устранить многие замечания, которые заявлялись пользователями. Однако и poetry набирает сейчас популярность, авторы устраняют найденные замечания и повышают надежность.

*Тест*

1. Отметьте утилиты для создания виртуальных окружений (0.25)
   1. virtual-environment
   2. **venv**
   3. windows
   4. **poetry**
2. Какой командой запускается виртуальное окружение? (0.25)
   1. start env
   2. **locate /bin/activate**
   3. run env
   4. activate env
3. Какой программой остановить виртуальное окружение? (0.25)
   1. stop
   2. release
   3. **deactivate**
   4. exit
4. Какая команда poetry добавить библиотеку numpy в проект? (0.25)
   1. **poetry add numpy**
   2. poetry install numpy
   3. poetry load numpy
   4. poetry get numpy

*Итоги/выводы*

В этом юните мы рассмотрели виртуальные программные окружения. Ознакомились с задачами, которые они решают. Изучили инструменты для их реализации.

# Модуль 3. Юнит 4. Основы контейнеризации с docker. Установка и настройка.

*Введение:* В этом юните вы познакомитесь с docker, изучите основные понятия, связанные с docker, научитесь устанавливать и настраивать docker.

*Содержание юнита:*

Этот юнит начинает ваше знакомство с одним из самых популярных инструментов для контейнеризации, docker. В этом юните вы познакомитесь с основными понятиями docker, его архитектурой. Также вы научитесь устанавливать и настраивать docker для работы.

**Контейнеризация приложения это упаковка приложения в отдельный контейнер, специальную среду с операционной системой и всеми необходимыми библиотеками, связями, зависимостями.** Контейнеризация приложений очень популярна в разработке программного обеспечения и практических задачах: организации разработки и тестирования, развертывании инфраструктуры распределенных систем, эксплуатации. Контейнеризация помогает сделать приложения более безопасными, облегчает их простое и быстрое развёртывание в продуктивной среде, улучшает масштабирование. Технология контейнеризации переживает фазу бурного роста и считается одним из основных направлений развития в индустрии разработки программного обеспечения.

**Технология Docker предназначена для разработки, установки и запуска в специальных сущностях - контейнерах.** С использованием инструментов Docker контейнеризуются программные продукты, информационные системы, отдельные приложения или масштабные системы со сложной архитектурой, состоящей из множества сервисов, в соответствии с концепцией микросервисной архитектуры.

Технология Docker настолько распространена, что стала фактически стандартом де-факто в контейнеризации, часто используется термин **«докеризация»**.

Технология контейнеризации идеологически похожа на виртуализацию, но есть существенные различия. Виртуальные машины обеспечивают полный уровень изоляции гостевых операционных систем, однако на это расходуется много ресурсов. Принцип работы Docker отличается от принципов работы виртуальных машин тем, что виртуальная машина взаимодействует напрямую с аппаратным обеспечением, а Docker работает с низкоуровневыми инструментами основной операционной системы, позволяя экономить ресурсы и выполнять задачи быстрее. На практике распространено применение docker -контейнеров в виртуальных машинах.

Давайте познакомимся с ключевыми понятиями и терминами docker.

**Docker Платформа (Docker Platform)** это программный комплекс, который упаковывает приложения в контейнеры, запускает контейнеры в аппаратных средах (серверах), управляет логикой работы контейнеров, обеспечивает работу пользователя в системе. Платформа Docker позволяет помещать в контейнеры код и его зависимости (используемые внешние библиотеки, переменные среды окружения, служебные файлы, параметры). При таком подходе упрощается запуск, перенос, воспроизведение, масштабирование систем.

**Docker «Движок» (Docker Engine)** это клиент-серверное приложение, обеспечивающее весь цикл работы с технологией Docker. Docker Engine может использоваться в одном из двух вариантов:

* Docker Community Edition - это бесплатное ПО, основанное на  инструментах open-source,
* Docker Enterprise – платное программное обеспечение, предназначенное для использования производственными компаниями в больших коммерческих проектах.

**Docker Клиент (Docker Client)** это основной инструмент пользователя при работе с Docker. Взаимодействие осуществляется с использованием командной строки Docker CLI (Docker Command Line Interface). В Docker CLI пользователь вводит команды, начинающиеся с ключевого слова «docker», эти команды обрабатываются Docker Клиентом и с использованием API Docker отправляются Docker Демону (Docker Daemon).

**Docker Образ (Docker Image)** это набор данных, содержащий:

* образ базовой операционной системы (файловая система, системные настройки, драйверы устройств),
* прикладное программное обеспечение для развертывания в базовой операционной системе, с настройками,
* библиотеки, служебные файлы.

Docker образ используется для создания **Docker Контейнера (Docker Container)**. Различают базовые и дочерние образы:

* *Base images (базовые образы)* не имеют родительского образа. Обычно это образы с операционной системой, такие как ubuntu, busybox или debian.
* *Child images (дочерние образы)* построены на базовых образах и обладают дополнительной функциональностью.

Существуют официальные и пользовательские образы (любые из них могут быть базовыми и дочерними):

* Официальные образы официально поддерживаются компанией Docker. Обычно в их названии одно слово (например, python, ubuntu).
* Пользовательские образы создаются пользователями и построены на базовых образах. Формат имени пользовательского образа «имя пользователя»/«имя образа».

**Docker Контейнер (Docker Container)** это запускаемый экземпляр Docker Образа. В контейнере запускаются приложения со всеми требуемыми настройками и зависимостями. Контейнер разделяет имеющиеся ресурсы на уровне ядра операционной системы с другими контейнерами, работает изолированно от других контейнеров в своей операционной системе (hosted OS).

**Docker Демон (Docker Daemon)**это сервис, запущенный в фоновом режиме, предназначенный для управления образами, контейнерами, сетями и томами. Взаимодействие с Docker Демоном осуществляется через запросы к API Docker.

**Docker Реестр, Docker хаб (Docker Hub)** это место хранения Образов Docker, облачное хранилище. Многие провайдеры услуг хостинга предоставляют возможность хранить образы Docker в своих репозиториях (например, Amazon, Yandex). Самым популярным и наиболее используемым хранилищем является официальный репозиторий [*Docker Hub*](https://hub.docker.com/) *(https://hub.docker.com/)*, используемый при работе с Docker по умолчанию. Обычно в репозиториях хранятся разные версии одних и тех же образов, обладающих одинаковыми именами и разными тегами (идентификаторами образов), разделенные двоеточием. Например,

* «python» - официальный репозиторий Python на Docker Hub,
* «python:3.7-slim» - версия образа с тегом «3.7-slim» в репозитории Python.

В реестр можно отправить как целый репозиторий, так и отдельный образ.

**Файл Dockerfile** это текстовый файл, содержащий упорядоченный перечень команд, необходимых при создании (building) Docker образа (Docker image). Этот файл содержит описание базового образа, который будет представлять собой исходный слой образа. Популярные официальные базовые образы:

|  |  |  |
| --- | --- | --- |
| alpine | легкая ОС linux, оптимальна для простых задач или обучения, для большинства задач требует дополнительной установки пакетов | https://hub.docker.com/\_/alpine |
| ubuntu | ОС linux с большим набором библиотек и утилит | https://hub.docker.com/\_/ubuntu |
| nginx | самый популярный и очень функциональный web сервер | https://hub.docker.com/\_/nginx |
| python | ОС linux с установленным программным обеспечением для работы с Python | https://hub.docker.com/\_/python/ |

В образ контейнера поверх базового образа можно добавлять дополнительные слои в соответствии с инструкциями из Dockerfile. Если Dockerfile описывает образ, который планируется использовать для решения задач машинного обучения, то в нём могут быть инструкции для включения библиотек машинного обучения NumPy, Pandas и Scikit-learn. В образе может содержаться, поверх всех остальных, ещё один «тонкий» слой, содержащий программу, которую планируется запускать в контейнере.

Пример Dockerfile

**FROM python # определяет базовый образ**

**COPY test.py /apps/ # копирует test.py в директорию /apps образа**

**CMD ["python", "/apps/test.py"] # запускает выполнение файла test.py**

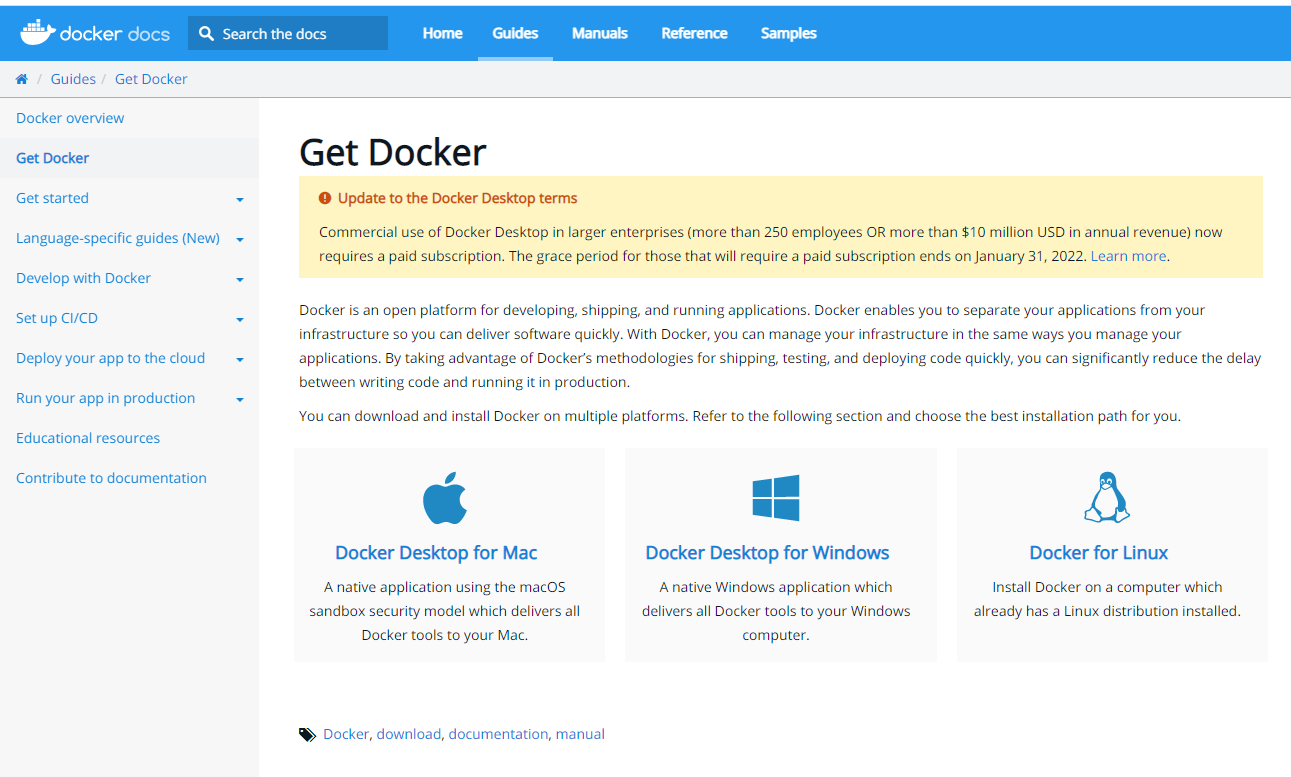
**Docker Том (Docker Volume)** это специальный уровень Docker контейнера, который позволяет передавать данные в Docker контейнер. Docker Том это наиболее предпочтительный механизм постоянного хранения данных, используемых или создаваемых приложениями. Здесь могут храниться таблицы базы данных, конфигурационные файлы, изображения, html файлы, сохраненные модели в формате pickle и т.п.

**Сетевые механизмы** **Docker (Docker Networking)** организуют связь между контейнерами Docker. Соединённые с помощью Docker Сети (Docker Network) контейнеры могут выполняться на одном и том же хосте или на разных хостах, взаимодействуя между собой как отдельные независимые сервисы.

**Docker Compose** это инструмент экосистемы Docker, упрощающий работу с многоконтейнерными приложениями. Docker Compose выполняет инструкции, описанные в файле docker-compose.yml.

Несмотря на то, что для Docker созданы инструменты для разных операционных систем, использование Windows в качестве хостовой операционной системы не рекомендуется, работать с такими образами сложнее. Изначально docker разрабатывался для linux операционных систем, например, Ubuntu, Red Hat, Debian. Поэтому работа с docker в linux операционных системах рекомендована как основной вариант использования. Давайте рассмотрим процедуру установки docker для операционной системы Ubuntu. Установка docker для различных операционных систем подробно описана здесь <https://docs.docker.com/engine/install/>.

Для установки docker для операционной системы Ubuntu необходимо на странице <https://docs.docker.com/get-docker/>



выбрать вариант “Docker for Linux”. После этого переходим к разделу “Server” в котором описаны варианты установки для разных версий операционных систем.

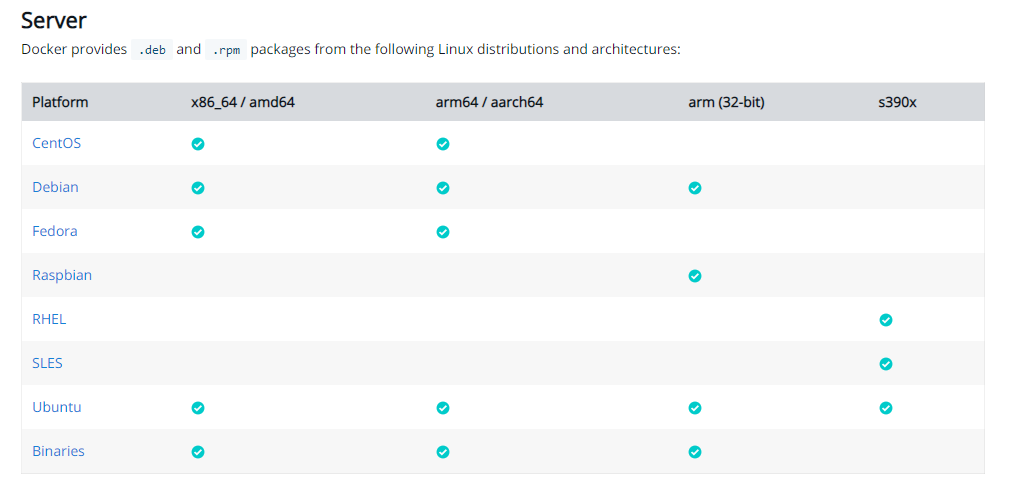


Рисунок “Техническая информация о возможностях использования docker для разных linux операционных систем”.

Поскольку мы решили устанавливать docker для операционной системы Ubuntu, необходимо выбрать соответствующий пункт таблицы после чего появится инструкция для установки.

Надо обратить внимание на необходимые параметры операционной системы, в которую мы будем устанавливать docker. На момент подготовки этого материала для установки Docker Engine требовалась одна из 64-битовых версий Ubuntu:

* Ubuntu Impish 21.10
* Ubuntu Hirsute 21.04
* Ubuntu Focal 20.04 (LTS)
* Ubuntu Bionic 18.04 (LTS)

В linux вы можете получить информацию о характеристиках операционной системы с помощью команды

**sudo uname -a**

После того как мы убедились в правильности версии операционной системы можно переходить к следующим шагам установки.

Сначала необходимо удалить старые версии docker, установленные в системе

**sudo apt-get remove docker docker-engine docker.io containerd runc**

Затем обновить установщик apt-get

**sudo apt-get update**

и установить необходимые пакеты для возможности использования HTTPS для установки

**$ sudo apt-get install ca-certificates curl gnupg lsb-release**

После этого добавляется ключ для взаимодействия с репозиторием docker по ssh

**$curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg**

и устанавливаются необходимые для этого настройки

**$echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \**

**$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null**

Для непосвященных это может казаться магией, но эти действия описаны в официальной инструкции docker по установке. При желании вы можете глубже погрузиться в этот синтаксис и разобрать действия описанных выше команд.

Теперь у нас все готово для установки, для этого еще раз обновим установщик apt-get и запустим установку с помощью команды apt-get install

**$sudo apt-get update**

**$sudo apt-get install docker-ce docker-ce-cli containerd.io**

После этого у нас docker установлен, в этом можно убедиться, выполнив команду в командной строке

**$docker**

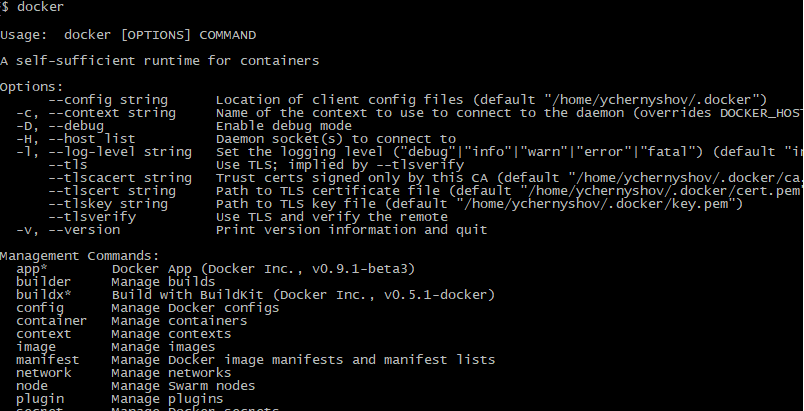
****

Рисунок “Основное окно docker”

Однако при попытке воспользоваться возможностями docker на данном этапе вы потерпите неудачу, которая будет выглядеть вот так



Проблема заключается в том, что обычный пользователь по умолчанию не имеет доступа к служебному сокету unix:///var/run/docker.sock через который идет взаимодействие. По умолчанию запуск команд docker требует прав суперпользователя и команды должны запускаться в формате “sudo docker …”. Для корректной работы необходимо настроить linux группы, имеющие специальные права доступа к служебным ресурсам Docker.

После установки docker в операционной системе уже существует специальная группа docker в которую можно добавлять пользователей. Проверить наличие группы можно в файле /etc/group. Если группы нет, то ее нужно создать

**sudo groupadd docker**

Добавить пользователя в группу

**sudo usermod –aG docker user**

Проверить какие пользователи входят в группу docker

**sudo members docker**

Удалить пользователя из группы

**sudo gpasswd –d user group**

**sudo deluser user group**

Для применения настроек пользователя необходимо зайти в систему. После того как группа создана и необходимый пользователь в нее добавлен, от имени этого пользователя можно выполнять команды docker.

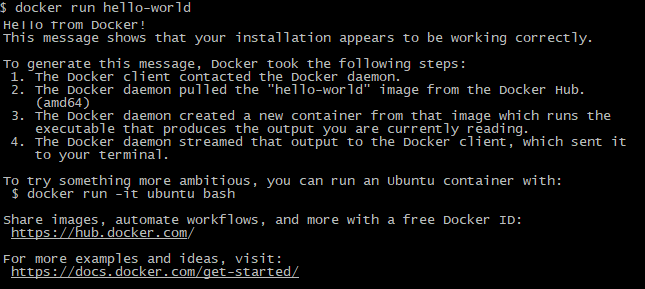


Рисунок “Результат выполнения команды docker run hello-docker”

При выполнении команды **docker run hello-world** выполняется запрос docker-образа hello-world:latest в реестре Docker, этот образ загружается на локальный Docker Engine, запускается в контейнере, в результате выполнения выводится сообщение «Hello from Docker!»

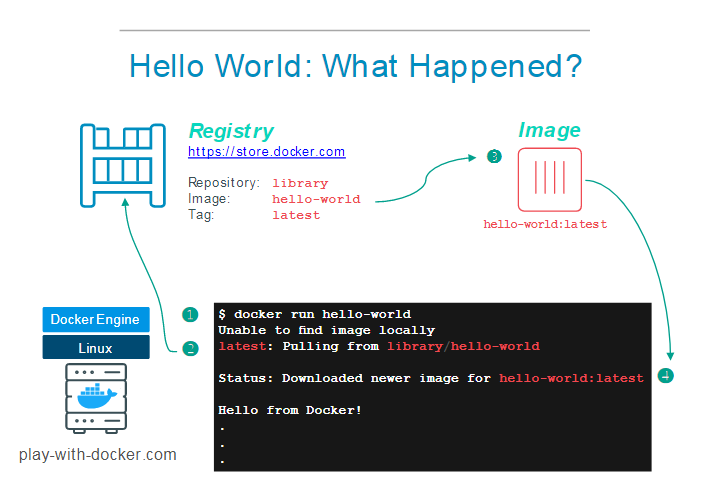


Рисунок “Hello World для Docker, пример с официального сайта docker.com”

*Тест*

1. Что такое docker образ (0.25)
   1. виртуальная машина
   2. **набор данных, содержащий операционную систему, служебные файлы, библиотеки, прикладное программное обеспечение**
   3. набор команд docker для создания контейнера
   4. исходный код docker
2. Что необходимо для запуска docker контейнера (0.25)
   1. программист
   2. **docker образ**
   3. доступ к сайту docker.org
   4. linux
3. В каком файле находятся команды для создания образа docker (0.25)
   1. commands-docker-file
   2. **Dockerfile**
   3. docker-instructions
   4. docker\_image\_data
4. Как избежать постоянного применения sudo при использовании docker (0.25)
   1. sudo docker unset
   2. **добавить пользователя в группу docker, обладающую соответствующими правами**
   3. sudo docker set primary mode
   4. внести соответствующие корректировки в Dockerfile

*Итоги/выводы*

В этом юните вы рассмотрели ключевые понятия и терминологию docker. В следующем юните вы изучите базовые команды docker для выполнения типовых операций.

# Модуль 3. Юнит 5. Базовые команды docker.

*Введение:* В этом юните вы узнаете базовые команды docker для выполнения типичных операций: создание и изменение образа, запуск контейнера, мониторинг работы.

*Содержание юнита:*

Этот юнит продолжит ваше знакомство с одним из самых популярных инструментов для контейнеризации, docker. В этом юните вы познакомитесь с основными командами docker и примерами применения docker для решения задач.

Базовые команды docker:

|  |  |
| --- | --- |
| docker --version | Используемая версия docker в сжатом формате |
| docker version | Расширенная информация о версиях различных компонентов docker |
| docker info | Информация о текущем состоянии docker, служебная информация, статистика, настройки |
| docker container run hello-world | Проверка работы docker («hello, world» для docker) |
| docker ps –a | Посмотреть информацию о контейнерах |
| docker rm <Container ID> | Удалить контейнер |
| docker stop <Container ID> | Остановить docker контейнер |
| docker network | Работа с docker сетями |
| docker image pull alpine | Загрузить docker образ (docker image) операционной системы alpine |
| docker image ls | Посмотреть имеющиеся в локальной системе пользователя docker образы (docker image) |
| docker container ls –a | Посмотреть все имеющиеся docker контейнеры (docker container) |
| docker container run alpine ls –l | Запустить команду «ls -l» в docker контейнере с docker образом операционной системы alpine |
| docker container run –it alpine sh | Запустить docker контейнер с docker образом с операционной системой alpine в интерактивном режиме и открыть терминал sh |
| docker container start <ContainerId> | Запустить docker контейнер с идентификатором <ContainerId> |
| docker container exec <ContainerId> <Command> | Выполнить команду <Command> в docker контейнере с идентификатором <ContainerId> |
| docker container diff <ContainerId> | Посмотреть изменения, сделанные в docker контейнере с идентификатором <ContainerId> |
| docker run -d -P --name <имя контейнера> <имя образа> | Запустить образ в контейнере в фоновом режиме (флаг –d, detached mode), все внутренние порты контейнера сделать внешними открытыми и случайными (флаг – P), внутренние (т.е. относящиеся к приложению в контейнере) и внешние (т.е. те, через которые контейнер общается в внешней средой) порты связываются. |
| docker ps –a –q | Вывести идентификаторы всех существующих контейнеров |
| docker rm $(docker ps –a –q –f status=exited) | Удалить все контейнеры, находящиеся в статусе Exited |
| docker port <Container ID> | Посмотреть порты, относящиеся к контейнеру |
| docker image build –t image\_name:v1 . | Создать новый образ с тэгом image\_name:v1. Последний параметр (точка, «.») указывает на то, что действия происходят в текущей директории (в которой должен находится Dockerfile) |
| docker container commit <ContainerId> | Создать новый образ на основе измененного контейнера |
| docker image tag <ContainerId> <Image Name> | Создать тег для образа |
| docker image history <ContainerId> | Посмотреть историю образа |
| docker image inspect <ImageName> | Просмотр детализированной информации об образе |
| docker image inspect –format “{{ json .Os }}” alpine | Просмотр детализированной информации об образе |
| docker image inspect –format “{{ json .RootFS.Layers }}” alpine | Просмотр детализированной информации об образе |
| docker push <Image name> | Загрузить образ в docker репозиторий |
| docker network create <имя сети> | Создать сеть docker для обмена сообщениями между контейнерами |
| docker run -n <имя сети> <имя контейнера> | Запустить контейнер с привязкой к сети |

**Основное полезное содержание работы с docker состоит в создании docker образов, включая операционную систему, необходимые библиотеки и утилиты и в последующем запуске контейнера на основе этого образа.** Рассмотрим далее эти две важные операции: создание образа и запуск контейнера.

**Новый Docker образ (Docker Image)** создается командой **docker build** с использованием инструкций в Dockerfile, при этом подразумевается, что Dockerfile находится в текущей рабочей директории. Если Dockerfile находится в другом месте, то его на расположение нужно указать с использованием флага -f. В файлах Dockerfile содержатся следующие инструкции по созданию образа:

|  |  |  |
| --- | --- | --- |
| Команда | Назначение | Пример использования |
| FROM | задаёт базовый (родительский) образ | FROM ubuntu |
| LABEL | описывает метаданные, например сведения о том, кто создал и поддерживает образ | LABEL maintainer="researcher1" |
| ENV | устанавливает постоянные переменные среды | ENV PATH=”/home/user” |
| RUN | выполняет команду и создаёт слой образа, используется для установки в контейнер пакетов | RUN pip install numpy |
| COPY | копирует в контейнер файлы и папки | COPY . /apps |
| ADD | копирует файлы и папки в контейнер, может распаковывать tar-файлы (архивы) | ADD test.py /apps |
| CMD | описывает команду с аргументами, которая должна выполниться при запуске контейнера, может быть лишь одна инструкция CMD | CMD [“python”, “test.py”] |
| WORKDIR | задаёт рабочую директорию для следующей за ней инструкции | WORKDIR /apps |
| ARG | задаёт переменные для передачи Docker во время сборки образа | ARG my\_var=1 |
| ENTRYPOINT | предоставляет команду с аргументами для вызова во время выполнения контейнера, аргументы не переопределяются. | ENTRYPOINT ["python", "./script.py"] |
| EXPOSE | указывает на необходимость открыть порт | EXPOSE 8000 |
| VOLUME | создаёт точку монтирования для работы с постоянным хранилищем | VOLUME /my\_volume |

В том случае когда контейнеров много, управлять ими становится сложно. Представьте, что вам придется запустить десять контейнеров командой docker container run и к концу запуска десятого вы поймете, что первый уже не работает… Можно конечно использовать shell скрипт, автоматизирующий эту ручную работу, но более универсальным инструментом является применение docker-compose. **Docker Compose** это инструментальное средство, входящее в состав Docker. Оно предназначено для решения задач, связанных с развертыванием проектов, состоящих из нескольких независимых совместно работающих приложений. docker-compose позволяет запускать и контролировать работу многих контейнеров, описывать их взаимодействие между собой, перезапускать при необходимости аварийно завершившиеся контейнеры.

Установка docker-compose описана здесь: <https://docs.docker.com/compose/install/>

Команды docker-compose

|  |  |
| --- | --- |
| docker-compose build | создать все необходимые docker-образы |
| docker-compose up | запустить все docker-контейнеры |
| docker-compose down | остановить все docker-контейнеры |
| docker-compose logs -f [service name] | посмотреть log-файлы |
| docker-compose ps | посмотреть все работающие контейнеры |
| docker-compose exec [service name] [command] | выполнить команду в определенном контейнере-сервисе |
| docker-compose images | посмотреть все доступные docker-образы |

При вызове команды docker-compose ищется файл docker-compose.yml, содержащий необходимые инструкции для docker-compose. Пример такого файла:



Руководствуясь командами из этого файла docker-compose запустит три контейнера, nginx, db и web, используя соответствующие параметры.

*Тест*

1. Какая команда запускает docker контейнер с использованием образа Image1? (0.25)
   1. docker container exec Image1
   2. docker container create Image1
   3. **docker container run Image1**
   4. docker container start Image1
2. Что произойдет при выполнении команды “docker container run alpine ls –l”? (0.25)
   1. ошибка выполнения команды
   2. запустятся два контейнера: alpine и ls с опцией -l
   3. **запустится контейнер с операционной системой alpine и в нем будет выполнена команда просмотра текущей директории**
   4. будет создан контейнер run
3. Для чего нужна директива FROM в Dockerfile (0.25)
   1. чтобы указать папку из которой требуется перенести файлы в контейнер
   2. для указания местонахождения главного исполняемого файла проекта
   3. **для определения основного образа, на основе которого будет создаваться контейнер**
   4. для применения интернет-ссылок
4. Какая команда docker-compose запустит все контейнеры, описанные в docker-compose.yml? (0.25)
   1. docker-compose begin
   2. **docker-compose up**
   3. docker-compose start
   4. docker-compose run

*Итоги/Выводы*

В этом юните вы изучили базовые команды docker, а также познакомились с утилитой docker-compose, которая позволяет запускать несколько docker контейнеров. В следующем юните вы выполните практическое задание с использованием docker, чтобы на практике применить уже имеющиеся у вас знания.

# 

# Модуль 3. Юнит 6. Практический пример использования docker.

*Введение:* В этом юните вы примените на практике имеющиеся знания по docker. Вы научитесь собирать образ и запускать контейнер docker, а также анализировать статистику.

*Содержание юнита:*

Напомним, что если у пользователя недостаточно прав для выполнения инструкций docker, то эти инструкции необходимо выполнять с правами суперпользователя (superuser), например

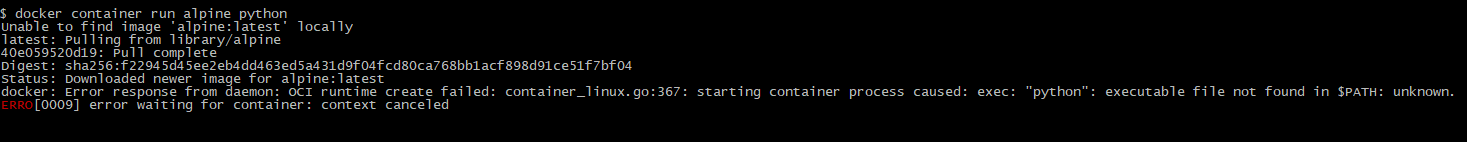
**sudo docker ps –a**

Такой подход не является рекомендованным, так как связан с повышенными рисками информационной безопасности, ведь пользователь может выполнять и другие действия с использованием sudo. Более правильная альтернатива использованию sudo это создание группы пользователей docker и наделение пользователей этой группы соответствующими правами. Далее команда sudo опускается.

Давайте создадим образ и запустим контейнер с этим образом. Для этого надо выполнить следующие шаги.

1. Сначала попробуем запустить docker-контейнер с “чистой” операционной системой alpine, запускающего простой python скрипт.

**docker container run alpine python**



По сообщениям системы видим, что python в образе alpine по умолчанию отсутствует. Это было предсказуемо, так как операционная система alpine по умолчанию содержит мало предустановленных утилит и библиотек, благодаря чему занимает очень мало места, но требует дополнительной настройки. По этой причине нам недостаточно контейнера с “чистой” операционной системой alpine и требуется дополнительно устанавливать необходимые нам служебные библиотеки и прикладное программное обеспечение.

1. Создаем файл Dockerfile с содержимым

**FROM alpine**

**RUN apk add python**

**COPY . /apps**

**WORKDIR /apps**

**CMD [“python”, “test.py”]**

1. Создаем файл test.py, который будет выполняться в контейнере, например

**a = [i\*\*2 for i in range(1,11)]**

**print(a)**

1. Создаем Docker образ

**docker image build -t test\_python:0.1 .**

Важно не забыть точку в конце этой конструкции, которая имеет важное значение, обозначает текущую директорию.

1. Запускаем контейнер с использованием созданного Docker образа

**docker container run test\_python:0.1**

1. Проверяем статус запущенного контейнера

**docker container ls -a**

Вы реализовали очень простой пример, не имеющий практической пользы. Дальше вы можете попробовать самостоятельно создать образ docker, содержащий необходимые версии библиотек программного обеспечения для проектов машинного обучения (например, numpy, pandas, scikit-learn) и выполнить в docker образе python код с обучением и использованием модели машинного обучения.

***Практическое задание***

Используя пример из юнита в качестве образца создайте docker образ для модели машинного обучения, в котором обучается модель логистической регрессии для предсказания типа цветка ириса из датасета sklearn.datasets.iris и возвращает предсказание класса на каком-то тестовом примере, например, [1, 1, 1, 1].

1. Подготовить python код для модели (0.25)
2. Создать Docker file (0.25)
3. Создать docker образ (0.25)
4. Запустить docker контейнер (0.25)

*Итоги/Выводы*

В этом юните вы выполнили практическое задание с использованием docker.

# Итоги/выводы по модулю

В этом модуле вы познакомились с технологиями создания изолированных сред, в которых могут выполняться программы: виртуализацией и контейнеризацией. Преимущества использования этих технологий:

* быстрое развертывание необходимой конфигурации
* удобное управление и мониторинг
* повышение уровня информационной безопасности, возможность локализовать вредоносные действия внутри изолированной среды, возможность быстро вернуться к работающей конфигурации
* единая техническая политика, упрощающая настройку и использование
* эффективное использование аппаратного обеспечения в режиме разделения времени, что повышает эффективность инвестиций в оборудование

# Практическое задание

В практическом задание по модулю вам необходимо применить полученные знания по работе с docker и docker-compose. Вам необходимо использовать полученные ранее знания по созданию микросервисов. В этом задании необходимо развернуть микросервис в контейнере докер на облачной платформе Яндекс. Например, это может быть модель машинного обучения, принимающая запрос по API и возвращающая ответ.

1. Получение доступа к инфраструктуре Яндекс.cloud.
   1. Создать аккаунт на yandex.ru.
   2. Зайти на cloud.yandex.ru <https://console.cloud.yandex.ru/>
2. Настройка окружения
   1. Создать виртуальную машину с публичным образом linux.
   2. Установить docker и docker-compose
3. Создать микросервисы с моделью машинного обучения, принимающей данные и отвечающей по API
4. Запуск через docker-compose

# Список терминов и сокращений

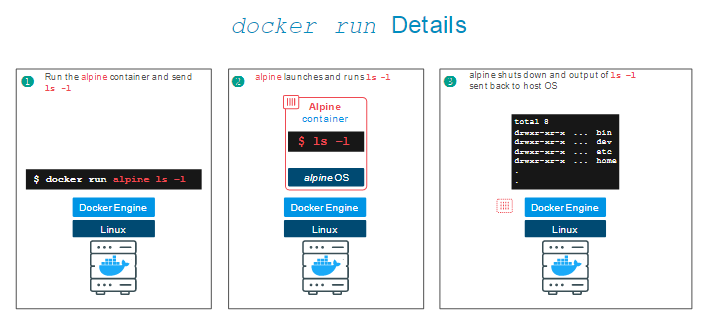
|  |  |
| --- | --- |
| ATA | Advanced Technology Attachment. Параллельный интерфейс подключения накопителей к компьютеру. |
| IDE | Integrated Drive Electronics. Параллельный интерфейс подключения накопителя (жесткого или оптического диска) к компьютеру или серверу. Маркетинговое название для ATA. |
| SATA | Serial ATA. Последовательный интерфейс обмена данными с накопителями информации. SATA является развитием параллельного интерфейса ATA (IDE). |

# Дополнительные материалы

В этом разделе приведены иллюстрации и описания основных команд docker, взятые с официальных ресурсов о docker в Интернет: [https://training.play-with-docker.com](https://training.play-with-docker.com/), docker.com и другие.

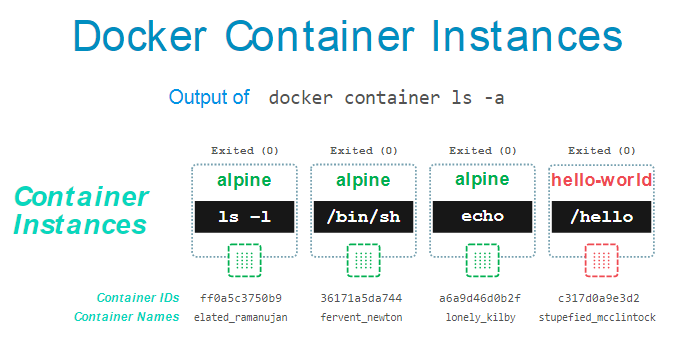
1. Выполнение команды docker run

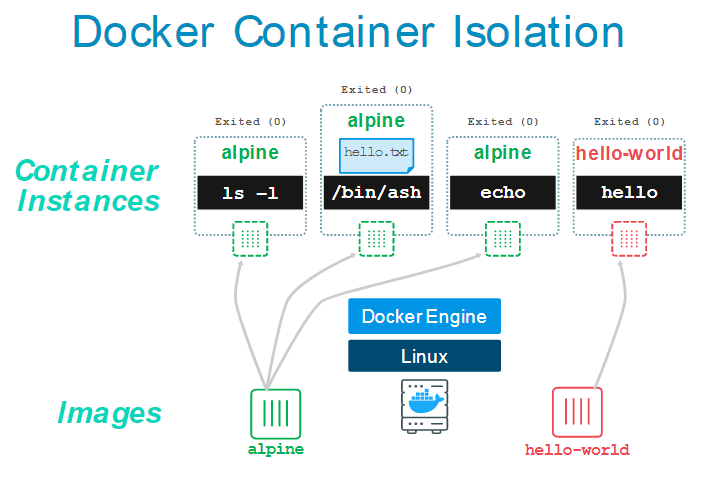
При выполнении команды **docker run <параметры> <имя образа> Docker Engine** запускает контейнер с указанным в команде образом и параметрами.



1. Docker контейнеры

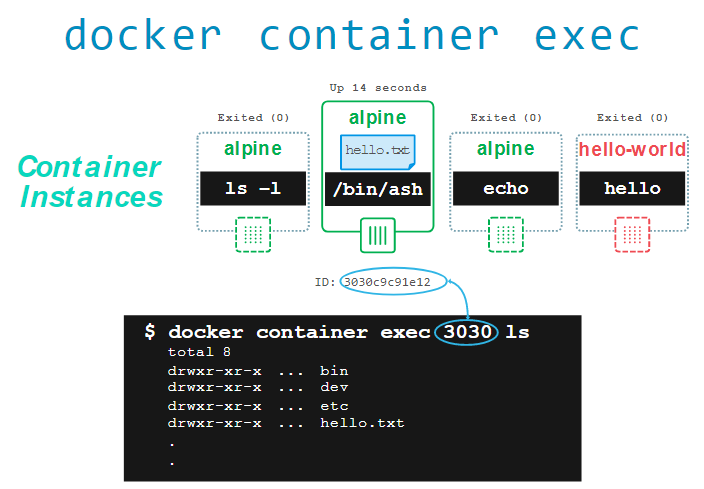
Docker контейнеры, запущенные на выполнение командой docker run, представляют собой раздельные изолированные сущности. У каждого контейнера есть свой идентификатор. Информацию о имеющихся контейнерах можно посмотреть командой **docker container ls -a** (флаг –a указывает на то, что надо посмотреть все контейнеры, а не только выполняющиеся)



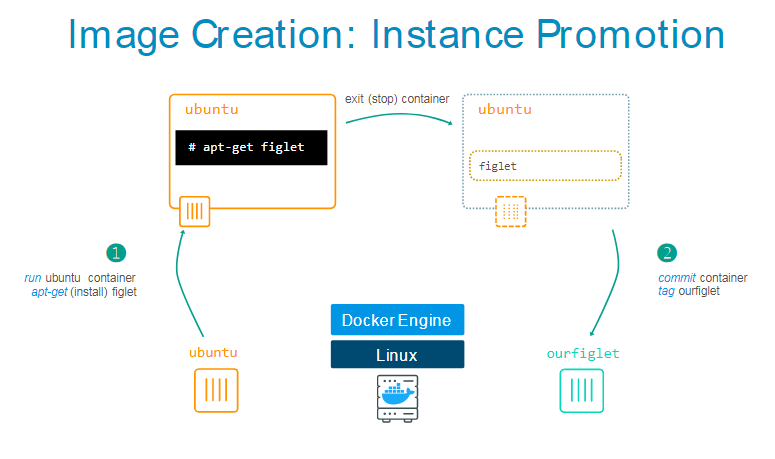


1. Выполнение команд в контейнере

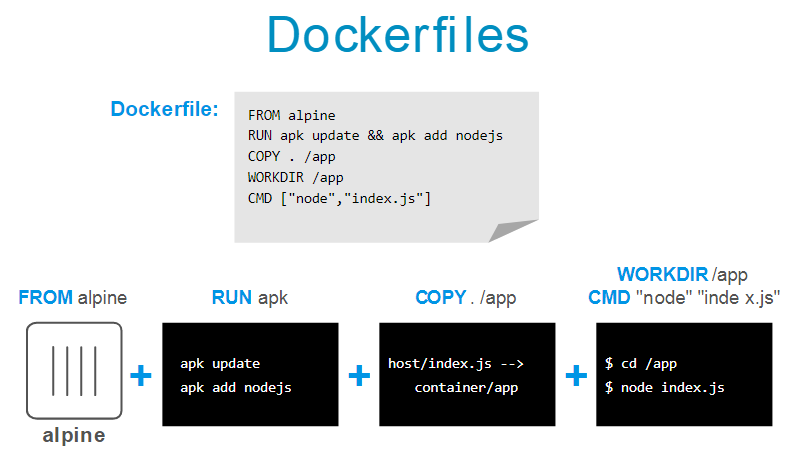
В отдельном контейнере можно выполнять команды, контейнеры идентифицируются номером.



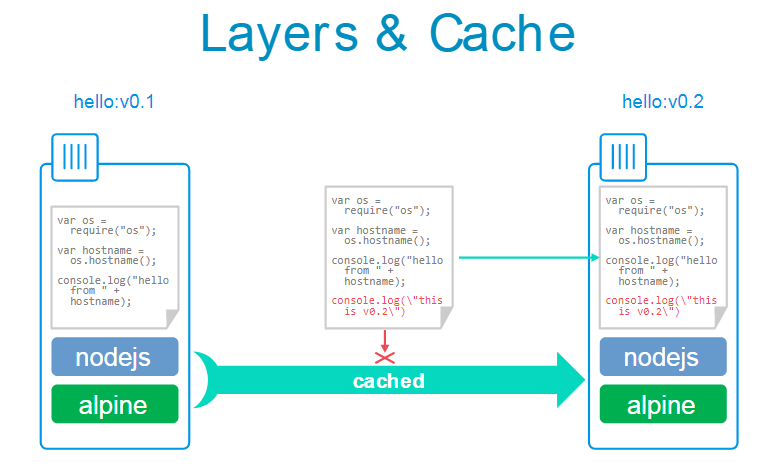
1. Запуск и остановка docker контейнера



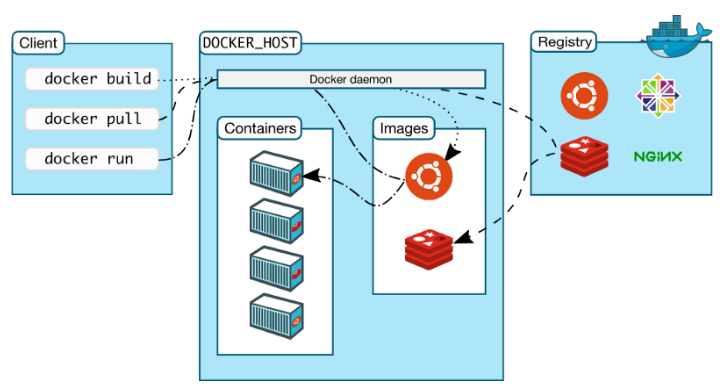
1. Dockerfile



1. Слои docker образа



1. Создание нового docker образа



# Ссылки

Курс «Инженерия машинного обучения»

https://lms.skillfactory.ru/courses/course-v1:SkillFactory+URFUML+SEP2021/about